L Number	Hits	Search Text	DB	Time stamp
1	103	("5805545" "5511053" "5583308" "6388181" "5792971" "5225618" "5256832" "5586967" "5689477" "5740260" "5914914" "6343055" "5726373" "5585583" "4794465" "4991032" "6185362" "4492990" "4755889" "4839746" "5606362" "5701153" "6055565" "6477134" "5739457" "6018121" "5675557" "5690496" "4339980" "3885490" "5313011" "4467287" "5530898" "4020291" "4916742" "5426540" "6282362" "5675511" "6236885" "5625410" "6025870" "4546690" "5889224" "4296664" "5215468" "6025553" "6184454" "4564930" "5825947" "5860679").pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:38
2	1887	345/716 or 345/717 or 345/718 or 345/719 or 345/720 or 345/721 or 345/723 or 345/726 or 345/733	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:39
3	1797	audio ADJ streams	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:40
4	3346	video ADJ streams	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:40

5	562	(audio ADJ streams) and events	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:41
6	1058	(video ADJ streams) and events	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:41
7	315	((audio ADJ streams) and events) and ((video ADJ streams) and events)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:41
8	1	(("5805545" "5511053" "5583308" "6388181" "5792971" "5225618" "5256832" "5586967" "5689477" "5740260" "5914914" "6343055" "5726373" "5585583" "4794465" "4991032" "6185362" "4492990" "4755889" "4839746" "5606362" "5701153" "6055565" "6477134" "5739457" "6018121" "5675557" "5690496" "4339980" "3885490" "5313011" "4467287" "5530898" "4020291" "4916742" "5426540" "6282362" "5675511" "6236885" "5625410" "6025870" "4546690" "5889224" "4296664" "5215468" "6025553" "6184454" "4564930" "5825947" "5860679").pn.) and (((audio ADJ streams) and events) and ((video ADJ streams) and events))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:41

9	15	(345/716 or 345/717 or 345/718 or 345/719 or 345/720 or 345/721 or 345/723 or 345/726 or 345/733) and (((audio ADJ streams) and events) and ((video ADJ streams) and events))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:42
10	15623	audio ADJ visual	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:42
11	4	((345/716 or 345/717 or 345/718 or 345/719 or 345/720 or 345/721 or 345/723 or 345/726 or 345/733) and (((audio ADJ streams) and events) and ((video ADJ streams) and events))) and (audio ADJ visual)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:42
12	0	(((345/716 or 345/717 or 345/718 or 345/719 or 345/720 or 345/721 or 345/723 or 345/726 or 345/733) and (((audio ADJ streams) and events) and ((video ADJ streams) and events))) and (audio ADJ visual)) and match	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:42
13	0	(((345/716 or 345/717 or 345/718 or 345/719 or 345/720 or 345/721 or 345/723 or 345/726 or 345/733) and (((audio ADJ streams) and events) and ((video ADJ streams) and events))) and (audio ADJ visual)) and compare	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:43
14	0	(((345/716 or 345/717 or 345/718 or 345/719 or 345/720 or 345/721 or 345/723 or 345/726 or 345/733) and (((audio ADJ streams) and events) and ((video ADJ streams) and events))) and (audio ADJ visual)) and synchronizing	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:43
15	6	((345/716 or 345/717 or 345/718 or 345/719 or 345/720 or 345/721 or 345/723 or 345/726 or 345/733) and (((audio ADJ streams) and events) and ((video ADJ streams) and events))) and matching	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:44
16	4362	reference ADJ location	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:45
17	4	audio ADJ tempo	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:45
18	0	(reference ADJ location) and (audio ADJ tempo)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:45
19	2	(audio ADJ tempo) and change	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:45
20	2	((audio ADJ tempo) and change) and reference	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:46
21	1	(((audio ADJ tempo) and change) and reference) and location	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:46

22	0	(((audio ADJ tempo) and change) and reference) and marked	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:47
23	0	(audio ADJ tempo) and partially	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:49
24	99	musical ADJ event	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:50
25	6	(audio ADJ streams) and (reference ADJ location)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:50
26	0	(musical ADJ event) and ((audio ADJ streams) and (reference ADJ location))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:50
27	0	((audio ADJ streams) and (reference ADJ location)) and musical	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:51
28	11212	video ADJ frame	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:52
29	1556	(audio ADJ streams) and reference	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:52
30	89	((audio ADJ streams) and reference) and coincide	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:53
31	26	(video ADJ frame) and (((audio ADJ streams) and reference) and coincide)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:53
32	26	((video ADJ frame) and (((audio ADJ streams) and reference) and coincide)) and time	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:53
33	19	(((video ADJ frame) and (((audio ADJ streams) and reference) and coincide)) and time) and location	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:53
34	17549	time ADJ reference	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:54
35	2	((((video ADJ frame) and (((audio ADJ streams) and reference) and coincide)) and time) and location) and (time ADJ reference)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:54

36	0	(((((video ADJ frame) and (((audio ADJ streams) and reference) and coincide)) and time) and location) and (time ADJ reference)) and synchronize	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:56
37	89	(((audio ADJ streams) and events) and ((video ADJ streams) and events)) and synchronizing	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:56
38	4	(time ADJ reference) and ((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and synchronizing)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:56
39	2	((time ADJ reference) and ((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and synchronizing)) and GUI	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:57
40	8347	graphical ADJ representations	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:57
41	0	(((time ADJ reference) and ((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and synchronizing)) and GUI) and (graphical ADJ representations)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:57
42	0	(((time ADJ reference) and ((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and synchronizing)) and GUI) and representations	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:58
43	816	audio ADJ waveform	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:59
44	0	(((time ADJ reference) and ((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and synchronizing)) and GUI) and (audio ADJ waveform)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 09:59
45	0	(((time ADJ reference) and ((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and synchronizing)) and GUI) and waveform	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:00
46	1	(((audio ADJ streams) and events) and ((video ADJ streams) and events)) and (graphical ADJ representations)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:00
47	0	((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and (graphical ADJ representations)) and waveform	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:00
48	15	(345/716 or 345/717 or 345/718 or 345/719 or 345/720 or 345/721 or 345/723 or 345/726 or 345/733) and (((audio ADJ streams) and events) and ((video ADJ streams) and events))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:01
49	0	(graphical ADJ representations) and ((345/716 or 345/717 or 345/718 or 345/719 or 345/720 or 345/721 or 345/723 or 345/726 or 345/733) and (((audio ADJ streams) and events) and ((video ADJ streams) and events)))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:01

50	4	((345/716 or 345/717 or 345/718 or 345/719 or 345/720 or 345/721 or 345/723 or 345/726 or 345/733) and (((audio ADJ streams) and events) and ((video ADJ streams) and events))) and GUI	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:01
51	0	(((345/716 or 345/717 or 345/718 or 345/719 or 345/720 or 345/721 or 345/723 or 345/726 or 345/733) and (((audio ADJ streams) and events) and ((video ADJ streams) and events))) and GUI) and waveform	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:02
52	104	(((audio ADJ streams) and events) and ((video ADJ streams) and events)) and GUI	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:02
53	0	(graphical ADJ representations) and ((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and GUI)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:03
54	43	((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and GUI) and representations	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:03
55	0	(((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and GUI) and representations) and waveform	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:04
56	4	(345/716 or 345/717 or 345/718 or 345/719 or 345/720 or 345/721 or 345/723 or 345/726 or 345/733) and ((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and GUI)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:04
57	0	((345/716 or 345/717 or 345/718 or 345/719 or 345/720 or 345/721 or 345/723 or 345/726 or 345/733) and ((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and GUI)) and waveform	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:07
58	21	(((audio ADJ streams) and events) and ((video ADJ streams) and events)) and waveform	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:07
59	15	((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and waveform) and manipulation	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:08
60	0	(((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and waveform) and manipulation) and GUI	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:08
61	0	((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and waveform) and GUI	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:09
62	15	(((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and waveform) and manipulation) and graphical	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:09

63	8	((((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and waveform) and manipulation) and graphical) and representations	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:10
64	8	(((((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and waveform) and manipulation) and graphical) and representations) and duration	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:10
65	0	((((((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and waveform) and manipulation) and graphical) and representations) and duration) and lenght	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:11
66	8	((((((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and waveform) and manipulation) and graphical) and representations) and duration) and length	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:11
67	0	(((((((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and waveform) and manipulation) and graphical) and representations) and duration) and length) and increase	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:11
68	0	(((((((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and waveform) and manipulation) and graphical) and representations) and duration) and length) and adjust	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:12
69	8	(((((((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and waveform) and manipulation) and graphical) and representations) and duration) and length) and change	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:12
70	4	video ADJ thumbnails	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:13
71	0	(((audio ADJ streams) and events) and ((video ADJ streams) and events)) and (video ADJ thumbnails)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:13
72	13	(((audio ADJ streams) and events) and ((video ADJ streams) and events)) and thumbnails	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:14
73	193	time ADJ labels	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:15
74	0	((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and thumbnails) and (time ADJ labels)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:15
75	12	((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and thumbnails) and times	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:15
76	12	(((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and thumbnails) and times) and images	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:15

77	12	((((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and thumbnails) and times) and images) and video	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:16
78	12	(((((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and thumbnails) and times) and images) and video) and audio	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:16
79	12	((((((((audio ADJ streams) and events) and ((video ADJ streams) and events)) and thumbnails) and times) and images) and video) and audio) and times	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/14 10:16

United States Patent [19]

Baumgartner et al.

[11] Patent Number: 5,642,171

[45] Date of Patent: Jun. 24, 1997

[54] METHOD AND APPARATUS FOR SYNCHRONIZING AUDIO AND VIDEO DATA STREAMS IN A MULTIMEDIA SYSTEM

[75] Inventors: Donn M. Baumgartner; Thomas A. Dye, both of Austin, Tex.

[73] Assignee: Dell USA, L.P., Round Rock, Tex.

[21] Appl. No.: 255,604

[22] Filed: Jun. 8, 1994

[51] Int. Cl.[6] H04N 5/04

[52] U.S. Cl. 348/515; 395/806

[58] Field of Search 348/515; 395/154, 395/162–164; 345/122; 375/355; H04N 5/04, 5/12

[56] References Cited

U.S. PATENT DOCUMENTS

Re. 33,535	2/1991	Cooper .	
4,538,176	8/1985	Haji et al. .	
4,618,890	10/1986	Kouyama et al. .	
4,644,400	2/1987	Kouyama et al. .	
4,679,085	7/1987	Chapelle et al. .	
4,703,355	10/1987	Cooper .	
4,750,034	6/1988	Lem .	
4,851,909	7/1989	Noske et al. .	
5,170,252	12/1992	Gear et al. .	
5,420,801	5/1995	Dockter et al.	364/514 R
5,430,485	7/1995	Lankford et al.	348/515
5,471,576	11/1995	Yee	395/154

FOREIGN PATENT DOCUMENTS

2305278	12/1990	Japan	348/515

OTHER PUBLICATIONS

Nicolaou, Cosmos "An Architecture for Real-Time Multimedia Communication Systems"; IEEE Journal on Selected Areas in Communications, vol. 8, No. 3, Apr. 1990.

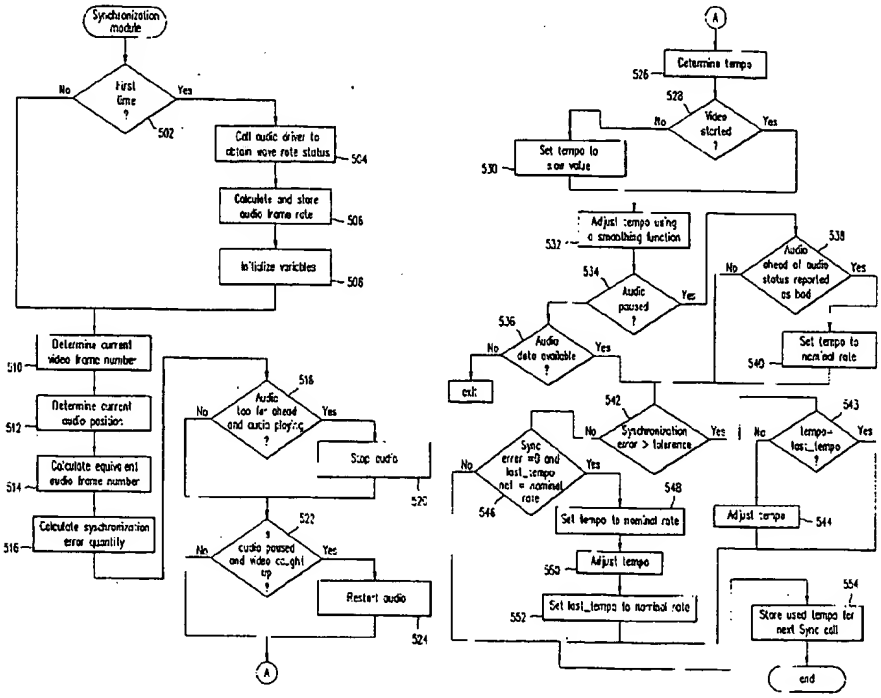
Little, Thomas D.C. and Arif Ghafoor "Synchronization and Storage Models for Multimedia Objects"; IEEE Journal on Selected Areas in Communications, vol. 8, No. 3, Apr. 1990.

Primary Examiner—Sherrie Hsia

[57] ABSTRACT

A method and apparatus for synchronizing audio and video data streams in a computer system during a multimedia presentation to produce a correctly synchronized presentation. The preferred embodiment of the invention utilizes a nonlinear feedback method for data synchronization. The method of the present invention periodically queries each driver for the current audio and video position (or frame number) and calculates the synchronization error. The synchronization error is used to determine a tempo value adjustment to one of the data stream designed to place the video and audio back in sync. The method then adjusts the audio or video tempo to maintain the audio and video data streams in synchrony. In the preferred embodiment of the invention, the video tempo is changed nonlinearly over time to achieve a match between the video position and the equivalent audio position. The method applies a smoothing function to the determined tempo value to prevent overcompensation. The method of the present invention can operate in any hardware system and in any software environment and can be adapted to existing systems with only minor modifications.

40 Claims, 7 Drawing Sheets



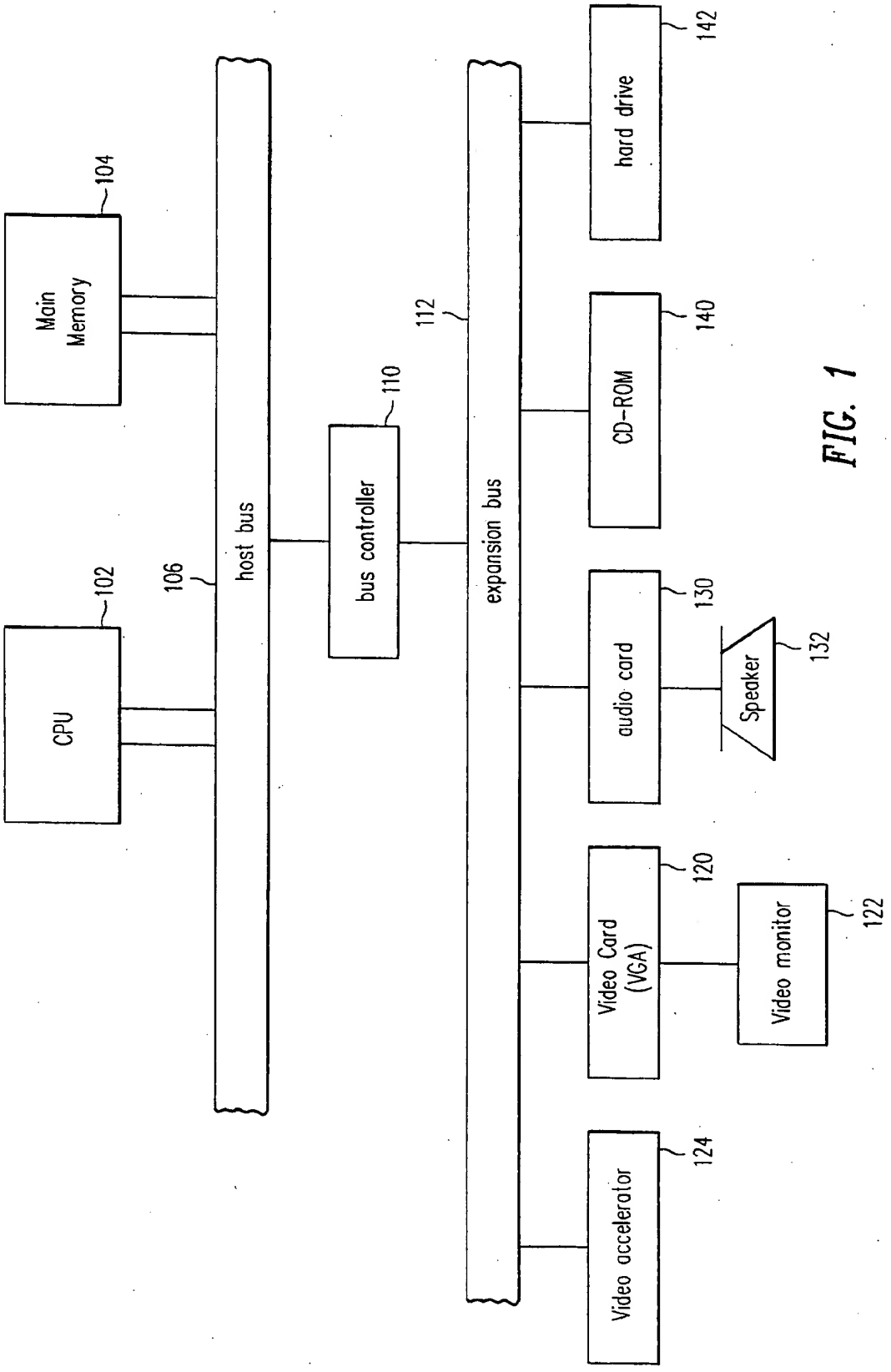

Main Memory
104
CPU
102
106
host bus
bus controller
110
112
expansion bus
hard drive
142
CD-ROM
140
audio card
130
Speaker
132
Video Card (VGA)
120
Video monitor
122
Video accelerator
124

FIG. 1

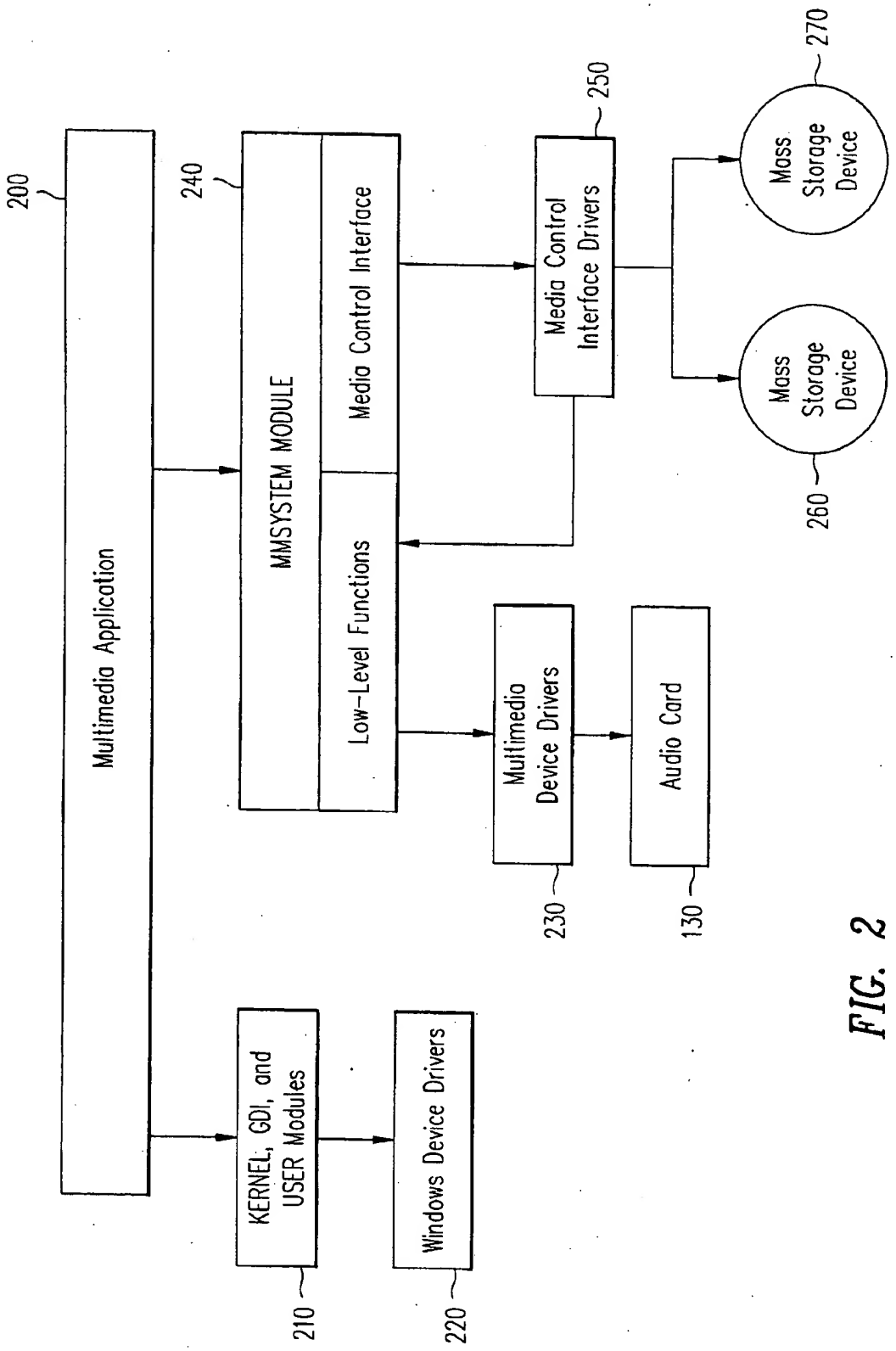

200
Multimedia Application
240
MMSYSTEM MODULE
Media Control Interface
Low-Level Functions
250
Media Control Interface Drivers
270
Mass Storage Device
260
Mass Storage Device
230
Multimedia Device Drivers
130
Audio Card
210
KERNEL, GDI, and USER Modules
220
Windows Device Drivers

FIG. 2

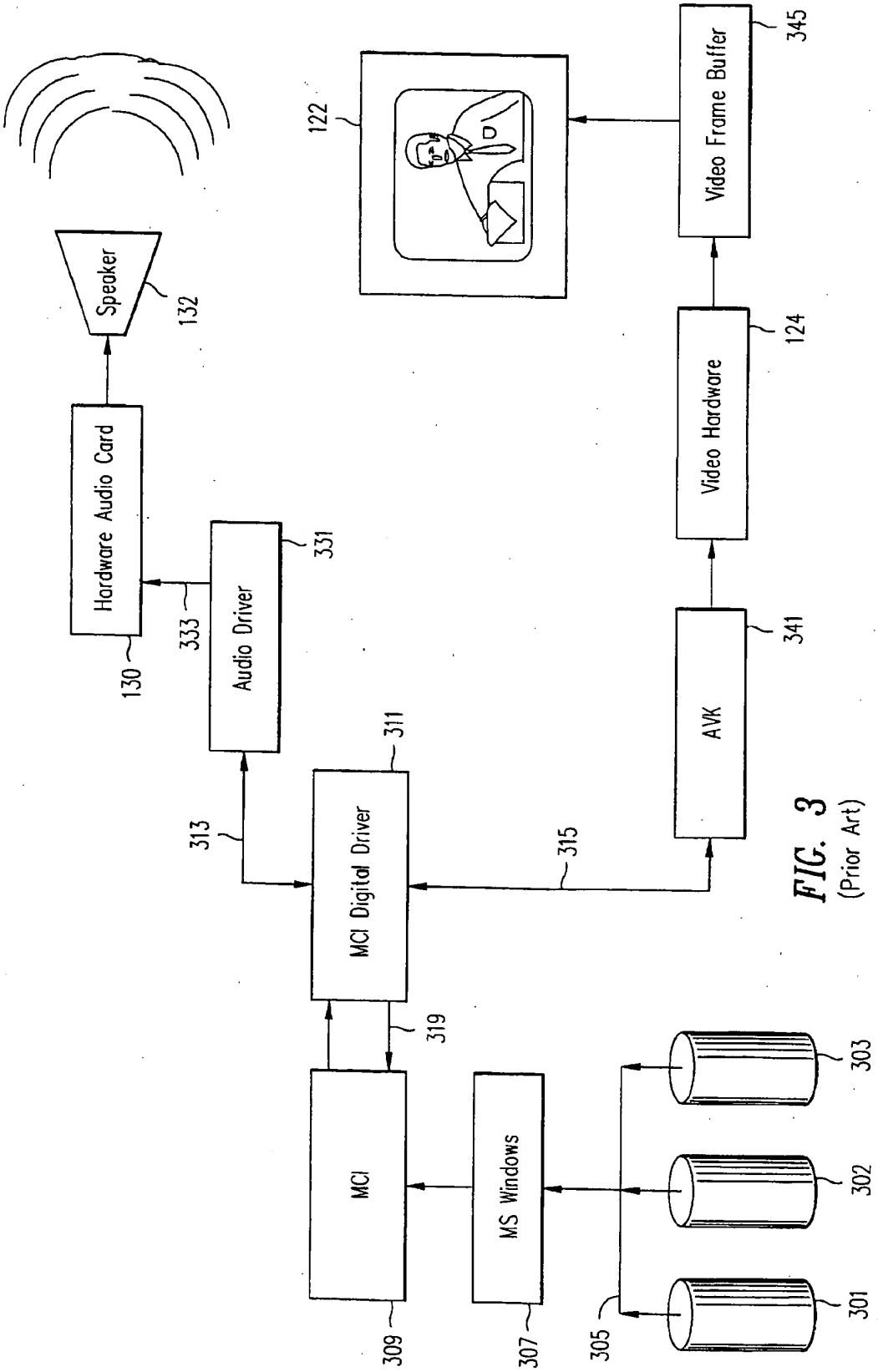


FIG. 3
(Prior Art)

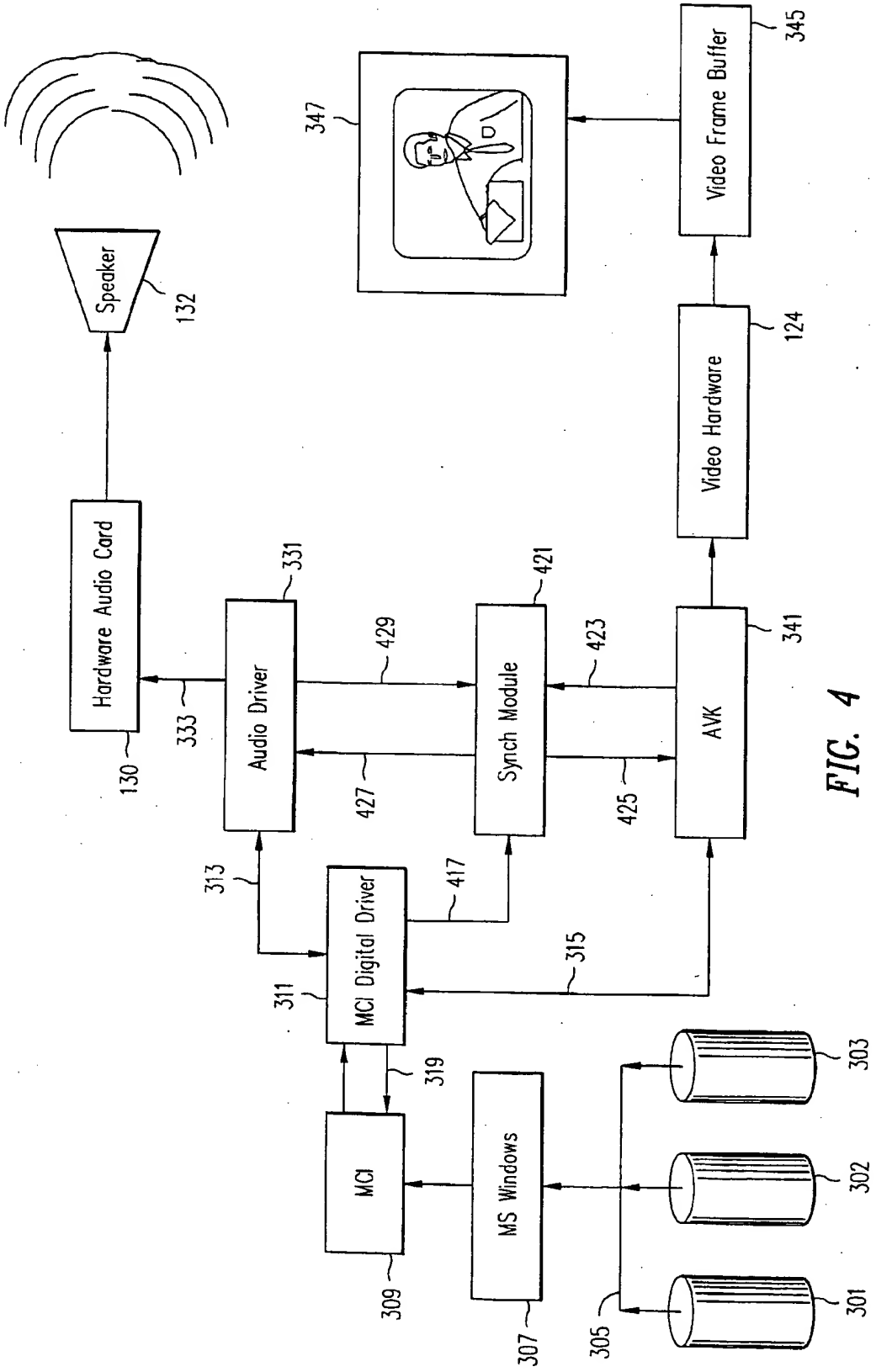

Speaker
132
Hardware Audio Card
130
333
Audio Driver
331
313
MCI Digital Driver
311
MCI
309
319
MS Windows
307
305
303
302
301
429
427
Synch Module
421
417
423
425
AVK
341
315
Video Hardware
124
Video Frame Buffer
345
347

FIG. 4

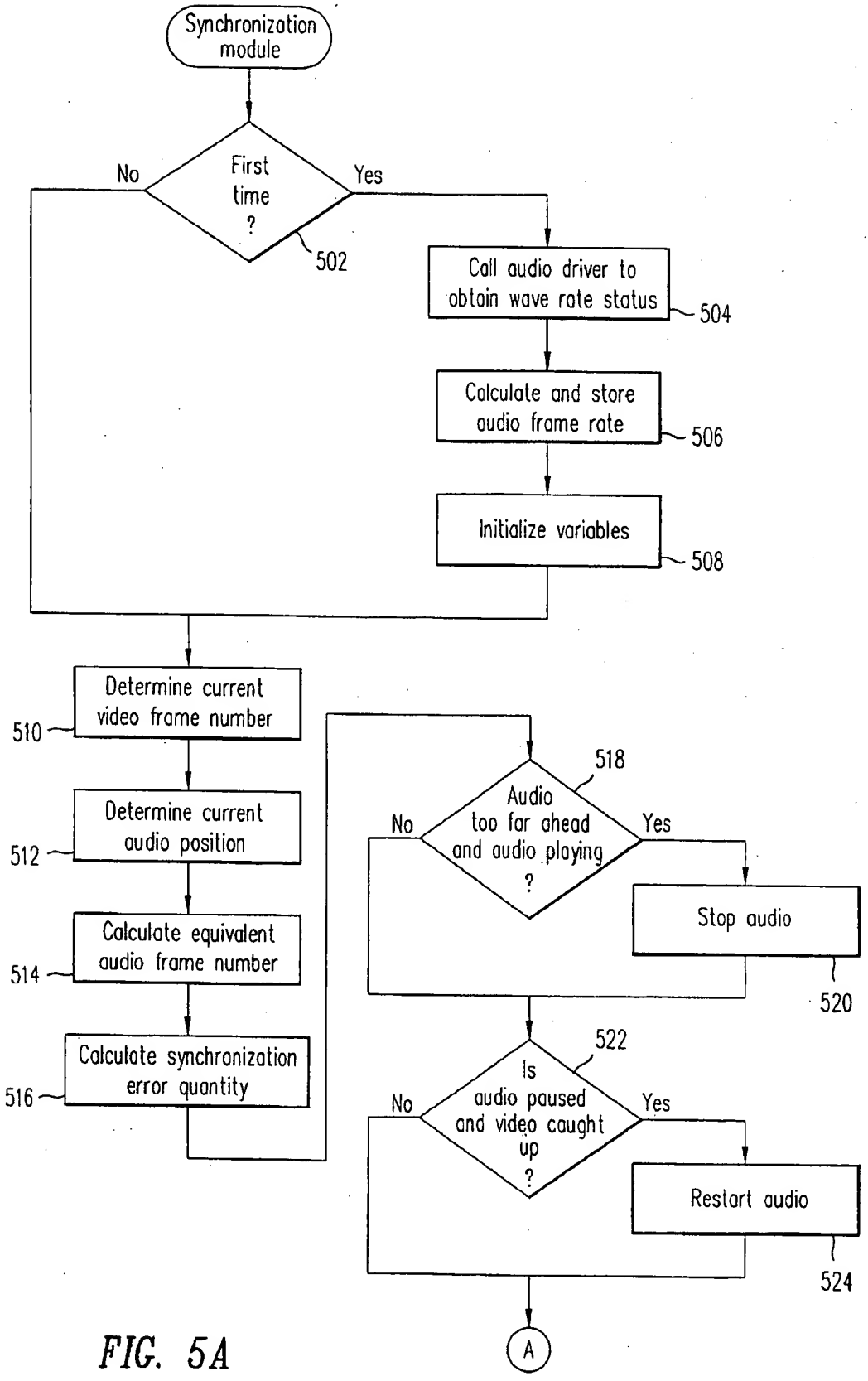
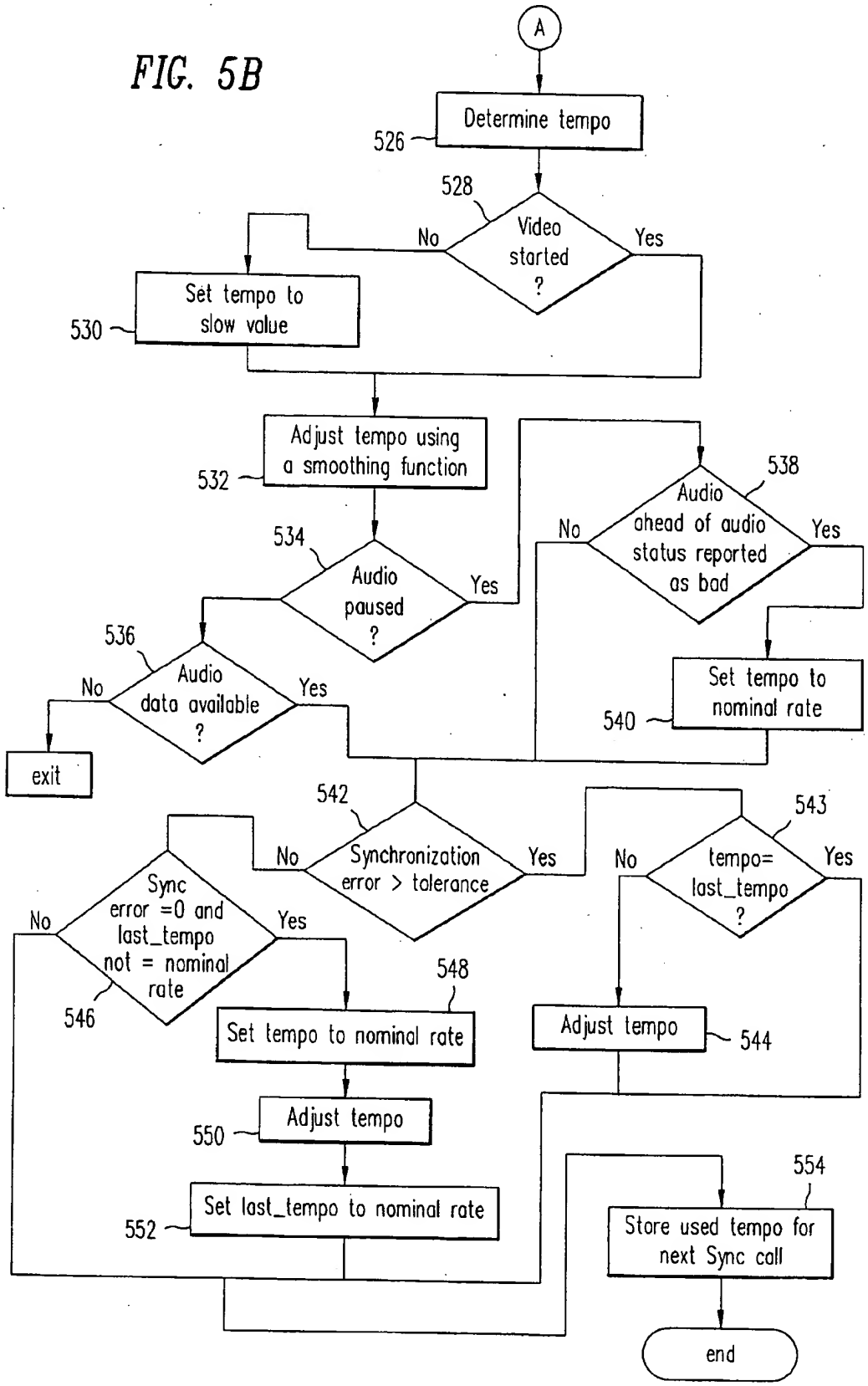

Synchronization module
First time ?
No
Yes
502
Call audio driver to obtain wave rate status
504
Calculate and store audio frame rate
506
Initialize variables
508
Determine current video frame number
510
Determine current audio position
512
Calculate equivalent audio frame number
514
Calculate synchronization error quantity
516
Audio too far ahead and audio playing ?
518
No
Yes
Stop audio
520
Is audio paused and video caught up ?
522
No
Yes
Restart audio
524
A

FIG. 5A

FIG. 5B


A
Determine tempo
526
528
Video started ?
No
Yes
Set tempo to slow value
530
Adjust tempo using a smoothing function
532
534
Audio paused ?
Yes
536
Audio data available ?
No
Yes
exit
538
Audio ahead of audio status reported as bad
No
Yes
Set tempo to nominal rate
540
542
Synchronization error > tolerance
No
Yes
543
tempo= last_tempo ?
No
Yes
Sync error =0 and last_tempo not = nominal rate
No
Yes
546
548
Set tempo to nominal rate
Adjust tempo
544
Adjust tempo
550
Set last_tempo to nominal rate
552
554
Store used tempo for next Sync call
end

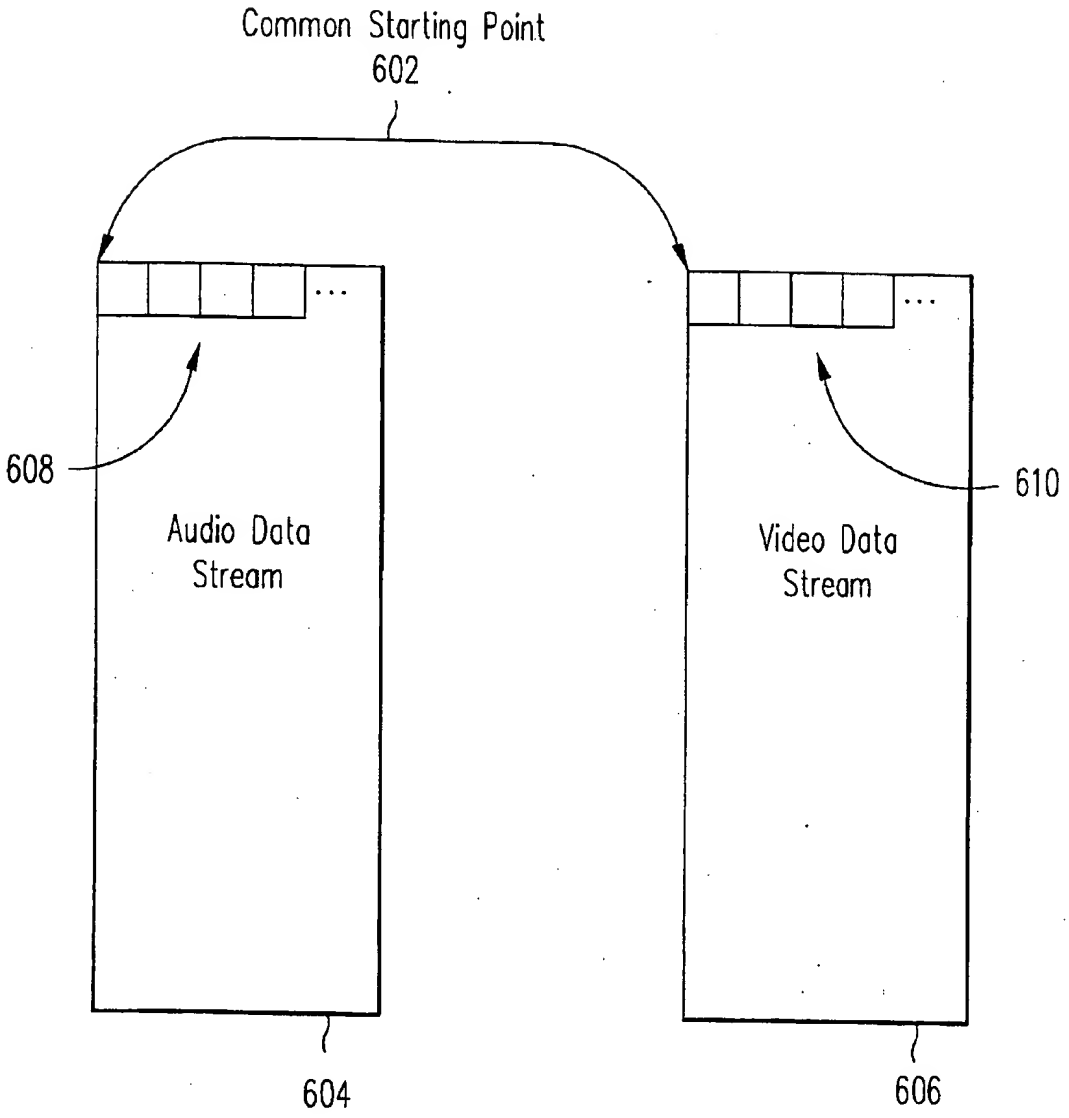

Common Starting Point
602
608
610
Audio Data
Stream
Video Data
Stream
604
606

FIG. 6

METHOD AND APPARATUS FOR SYNCHRONIZING AUDIO AND VIDEO DATA STREAMS IN A MULTIMEDIA SYSTEM

FIELD OF THE INVENTION

The present invention relates generally to multimedia computer systems, and more particularly to a method and apparatus for synchronizing video and audio data streams in a computer system during a multimedia presentation.

DESCRIPTION OF THE RELATED ART

Multimedia computer systems have become increasingly popular over the last several years due to their versatility and their interactive presentation style. A multimedia computer system can be defined as a computer system having a combination of video and audio outputs for presentation of audio-visual displays. A modern multimedia computer system typically includes one or more storage devices such as an optical drive, a CD-ROM, a hard drive, a videodisc, or an audiodisc, and audio and video data are typically stored on one or more of these mass storage devices. In some file formats the audio and video are interleaved together in a single file, while in other formats the audio and video data are stored in different files, many times on different storage media. Audio and video data for a multimedia display may also be stored in separate computer systems that are networked together. In this instance, the computer system presenting the multimedia display would receive a portion of the necessary data from the other computer system via the network cabling.

A multimedia computer system also includes a video card such as a VGA (Video Graphics Array) card which provides output to a video monitor, and a sound card which provides audio output to speakers. A multimedia computer system may also include a video accelerator card or other specialized video processing card for performing video functions, such as compression, decompression, etc. When a computer system displays a multimedia presentation, the computer system microprocessor reads the audio and video data stored on the respective mass storage devices, or received from the other computer system in a distributed system, and provides the audio stream through the sound card to the speakers and provides the video stream through the VGA card and any specialized video processing hardware to the computer video monitor. Therefore, when a computer system presents an audio-visual display, the audio data stream is decoupled from the video data stream, and the audio and video data streams are processed by separate hardware subsystems.

A multimedia computer system also includes an operating system and drivers for controlling the various hardware elements used to create the multimedia display. For example, a multimedia computer includes an audio driver or sound card driver for controlling the sound card and a video driver for controlling the optional video processing card. One example of an operating system which supports multimedia presentations is the Multimedia Extensions for the Microsoft Windows operating system.

Graphic images used in Windows multimedia applications can be created in either of two ways, these being bit-mapped images and vector-based images. Bit-mapped images comprise a plurality of picture elements (pixels) and are created by assigning a color to each pixel inside the image boundary. Most bit-mapped color images require one byte per pixel for storage, so large bit-mapped images create correspondingly large files. For example, a full-screen, 256-color image in 640-by-480-pixel VGA mode requires 307,

200 bytes of storage, if the data is not compressed. Vector-based images are created by defining the end points, thickness, color, pattern and curvature of lines and solid objects comprised within the image. Thus, a vector-based image includes a definition which consists of a numerical representation of the coordinates of the object, referenced to a corner of the image.

Bit-mapped images are the most prevalent type of image storage format, and the most common bit-mapped-image file formats are as follows. A file format referred to as BMP is used for Windows bit-map files in 1-, 2-, 4-, 8-, and 24-bit color depths. BMP files contain a bit-map header that defines the size of the image, the number of color planes, the type of compression used (if any), and the palette used. The Windows DIB (device-independent bit-map) format is a variant of the BMP format that includes a color table defining the RGB (red green blue) values of the colors used. Other types of bit-map formats include the TIF (tagged image format file), the PCX (Zsoft Personal Computer Paintbrush Bitmap) file format, the GIF (graphics interchange file) format, and the TGA (Texas Instruments Graphic Architecture) file format.

The standard Windows format for bit-mapped images is a 256-color device-independent bit map (DIB) with a BMP (the Windows bit-mapped file format) or sometimes a DIB extension. The standard Windows format for vector-based images is referred to as WMF (Windows meta file).

Compression

Full-motion video implies that video images shown on the computer's screen simulate those of a television set with identical (30 frames-per-second) frame rates, and that these images are accompanied by high-quality stereo sound. A large amount of storage is required for high-resolution color images, not to mention a full-motion video sequence. For example, a single frame of NTSC video at 640-by-400-pixel resolution with 16-bit color requires 512K of data per frame. At 30 flames per second, over 15 Megabytes of data storage are required for each second of full motion video. Due to the large amount of storage required for full motion video, various types of video compression algorithms are used to reduce the amount of necessary storage. Video compression can be performed either in real-time, i.e., on the fly during video capture, or on the stored video file after the video data has been captured and stored on the media. In addition, different video compression methods exist for still graphic images and for full-motion video.

Examples of video data compression for still graphic images are RLE (run-length encoding) and JPEG (Joint Photographic Experts Group) compression. RLE is the standard compression method for Windows BMP and DIB files. The RLE compression method operates by testing for duplicated pixels in a single line of the bit map and stores the number of consecutive duplicate pixels rather than the data for the pixel itself. JPEG compression is a group of related standards that provide either lossless (no image quality degradation) or lossy (imperceptible to severe degradation) compression types. Although JPEG compression was designed for the compression of still images rather than video, several manufacturers supply JPEG compression adapter cards for motion video applications.

In contrast to compression algorithms for still images, most video compression algorithms are designed to compress full motion video. Video compression algorithms for motion video generally use a concept referred to as inter-frame compression, which involves storing only the differ-

ences between successive frames in the data file. Interframe compression begins by digitizing the entire image of a key frame. Successive frames are compared with the key frame, and only the differences between the digitized data from the key frame and from the successive frames are stored. Periodically, such as when new scenes are displayed, new key frames are digitized and stored, and subsequent comparisons begin from this new reference point. It is noted that interframe compression ratios are content-dependent, i.e., if the video clip being compressed includes many abrupt scene transitions from one image to another, the compression is less efficient. Examples of video compression which use an interframe compression technique are MPEG, DVI and Indeo, among others.

MPEG (Moving Pictures Experts Group) compression is a set of methods for compression and decompression of full motion video images that uses the interframe compression technique described above. The MPEG standard requires that sound be recorded simultaneously with the video data, and the video and audio data are interleaved in a single file to attempt to maintain the video and audio synchronized during playback. The audio data is typically compressed as well, and the MPEG standard specifies an audio compression method referred to as ADPCM (Adaptive Differential Pulse Code Modulation) for audio data.

A standard referred to as Digital Video Interactive (DVI) format developed by Intel Corporation is a compression and storage format for full-motion video and high-fidelity audio data. The DVI standard uses interframe compression techniques similar to that of the MPEG standard and uses ADPCM compression for audio data. The compression method used in DVI is referred to as RTV 2.0 (real time video), and this compression method is incorporated into Intel's AVK (audio/video kernel) software for its DVI product line. IBM has adopted DVI as the standard for displaying video for its Ultimedia product line. The DVI file format is based on the Intel i750 chipset and is supported through the Media Control Interface (MCI) for Windows. Microsoft and Intel jointly announced the creation of the DV MCI (digital video media control interface) command set for Windows 3.1 in 1992.

The Microsoft Audio Video Interleaved (AVI) format is a special compressed file structure format designed to enable video images and synchronized sound stored on CD-ROMs to be played on PCs with standard VGA displays and audio adapter cards. The AVI compression method uses an interframe method, i.e., the differences between successive frames are stored in a manner similar to the compression methods used in DVI and MPEG. The AVI format uses symmetrical software compression-decompression techniques, i.e., both compression and decompression are performed in real time. Thus AVI files can be created by recording video images and sound in AVI format from a VCR or television broadcast in real time, if enough free hard disk space is available.

In the AVI format, data is organized so that coded frame numbers are located in the middle of an encoded data file containing the compressed audio and compressed video. The digitized audio and video data are organized into a series of frames, each having header information. Each frame of the audio and video data streams is tagged with a frame number that typically depends upon the frame rate. For example, at every 33 milliseconds (ms) or a 30th of a second, a frame number is embedded in the header of the video frame and at every 30th of a second, or 33 ms, the same frame number is embedded in the header of the audio track. The number assigned to the frames is, therefore, coordinated so that the corresponding audio and video frames are originally tagged with the same number. Therefore, since the frames are initially received simultaneously, the frames can actually be preprocessed so that tag codes are placed into the header files of the audio and the video for tracking the frame number and position of the audio and video tracks.

In the AVI format, the audio and video information are interleaved (alternated in blocks) in the CD-ROM to minimize delays that would result from using separate tracks for video and audio information. Also, the audio and video data are interleaved to synchronize the data as it is stored on the system. This is done in an attempt to synchronize the audio and video data during playback.

The Apple QuickTime format was developed by Apple for displaying animation and video on Macintosh computers, and has become a de facto multimedia standard. Apple's QuickTime and Microsoft's AVI take a parallel approach to the presentation of video stored on CD-ROMs, and the performance of the two systems is similar. The QuickTime format, like AVI, uses software compression and decompression techniques but also can employ hardware devices, similar to those employed by DVI, to speed processing. The Apple QuickTime format became available for the PC under Microsoft Windows in late 1992.

As mentioned above, the audio and video data streams in a multimedia presentation are processed by separate hardware subsystems under the control of separate device drivers. The audio and video data are separated into separate data streams that are then transmitted to separate audio and video subsystems. The video data is transmitted to the video subsystem for display, and the audio data is transmitted to the sound subsystem for broadcast. These two subsystems are addressed by separate drivers, and each driver is loaded dynamically by the operating system during a multimedia presentation. In an operating system that is multi-tasking, has multiple drivers, or has multiple windows, the time period between the servicing of drivers is indeterminate. If a driver is not serviced by the operating system in time for the next frame, a portion of the multimedia systems may stall, resulting in the audio not being synchronized with the video. When the audio and video portions of a multimedia presentation become unsynchronized, many times this lack of synchronization is noticeable to the viewer, resulting in a less pleasing display. One result of audio and video data being out of sync is that the viewer may hear words that do not match the lips of the speaker, a situation commonly called "out of lip sync."

Therefore, many times the corresponding audio and video frames of a multimedia presentation are not played synchronously together. The reasons for the audio and video data streams falling out of sync during a presentation include the inherent decoupling of the audio and video data streams in separate subsystems in conjunction with system bottlenecks and performance issues associated with the large amounts of data that are required to be manipulated during a multimedia presentation. As mentioned above, full motion video clips with corresponding audio require massive amounts of system resources to process. However, a considerably greater amount of processing is required to display the video data than is required for the audio data. First the video data must be decompressed either in software or in a codec (compression-decompression) device. If the color depth of the video is higher than that of the display, such as when an AVI file with 16 bit video is played on an 8 bit display, the computer must dither colors to fit within the display's color restrictions. Also, if the selected playback window size is inconsistent with the resolution at which the video was captured, the computer is required to scale each frame.

In addition to the greater amount of processing required for video data, the amount of video processing can vary considerably, thus further adversely affecting synchronization. For example, one variable that affects the speed of video playback is the decompression performed on the video data. The performance of software decompression algorithms can vary for a number of reasons. For example, due to the interframe method of compressing data, the number of bytes that comprise each video frame is variable, depending on how similar the prior video frame is to the current video frame. Thus, more time is required to process a series of frames in which background is moving than is required to process a series of frames containing only minor changes in the foreground. Other variables include whether the color depth of the video equals that of the display and whether the selected playback window size is consistent with the resolution at which the video was captured, as mentioned above.

In addition, a slow CPU adversely affects every stage in the processing of a video file for playback. A sluggish hard disk or CD-ROM controller can also adversely affect performance as can the performance of the display controller or video card. Also, other demands can be made on the system as a result of something as simple as a mouse movement. While the above processing is being performed on the video and audio data, and while other demands are made on system resources, it becomes very difficult to ensure that the audio and video data remain in synchronization.

Video for Windows includes a method which presumably attempts to maintain the audio and video portions of a multimedia display in sync, i.e., attempts to adapt when the computer system cannot keep up with either the video or audio portions of the display. Video for Windows benchmarks the video hardware when it first begins execution as well as every time thereafter that the default display is changed. The results of these tests are used to determine a particular system's baseline display performance at various resolutions and color depths. Video for Windows then uses this information regarding the capabilities of the video system to adjust the video frame rate to match the benchmarked performance for the default display. Video for Windows maintains the continuity of the audio at all costs because a halting audio track is deemed more distracting. When the burden of the video playback is such that the system cannot keep up, Video for Windows skips frames during playback or adjusts the frame rate continuously as the system's resource usage patterns change.

However, the method used by video for Windows in adjusting the video rate to match the benchmarked performance of the default display results in an average frame rate suitable for the benchmark determined at the time the default was last changed. Attempts to display video frames containing an unusually heavy amount of non-repetitive data will slow processing down to the point where the benchmarked frame rate is no longer useful. When this happens, video frames are skipped because the burden of processing the video data becomes too great to preserve lip-sync in the display. The result can be "jerky" movement of the images of persons speaking as noted in *Discover Windows* 3.1 *Multimedia*, by Roger Jennings (Que Corp. 1992), p. 105–106. Thus, the method used by Video for Windows has proven to be inadequate, i.e., the video and audio portions still fall out of sync or exhibit "jerky" movement during a presentation.

Shortcomings inherent in decoupled audio multimedia systems have been a problem for some time, and various efforts have been made to synchronize the audio and video portions of a presentation. There has been a recognized need in the industry for a solution to this problem. However, no satisfactory solution has been found, prior to the present invention.

Therefore, a method and apparatus is desired which provides improved synchronization between digital audio and digital video data streams in a multimedia computer system, i.e., a method is needed to assure that corresponding video and audio frames are played back together. A synchronization method is also desired that does not require the use of an encoding procedure prior to the processing of audio and video digital signals. It is also desirable to provide a multimedia synchronization system that is capable of functioning consistently whether video and audio data are delivered to the system in separate files or interleaved in one file.

SUMMARY OF THE INVENTION

The present invention comprises a method and apparatus for synchronizing separate audio and video data streams in a multimedia system. The preferred embodiment of the invention utilizes a nonlinear feedback method for data synchronization that is independent of hardware, the operating system and the video and audio drivers used. The system and method of the present invention does not require that incoming data be time stamped, or that any timing information exist in the video data stream relative to audio and video data correspondence. Further, the data is not required to be modified in any way prior to the transfer of data to the video and audio drivers, and no synchronization information need be present in the separated audio and video data streams that are being synchronized by the system and method of the present invention. The preferred embodiment of the present invention requires that there be a common starting point for the audio and video data, i.e., that them be a time index of zero where the audio and video are both in synchrony, such that the first byte of audio and video digital data are generated simultaneously.

The synchronization method of the present invention is called periodically during a multimedia display to synchronize the video and audio data streams. In the preferred embodiment, a periodic timer is set to interrupt the multimedia operating system at uniform intervals during a multimedia display and direct the operating system to invoke the synchronization method of the present invention. When the synchronization method is invoked, the method first queries the video driver to determine the current video frame position and then queries the audio driver to determine the current audio position. The current audio position is then used to compute the equivalent audio frame number. The synchronization method compares the video and audio frame positions and computes a synchronization error value, which is essentially the number of frames by which the video frame position is in front of or behind the current audio frame position.

The synchronization error is used to assign a tempo value meaningful to either the video driver or the audio driver. In the preferred embodiment, the method adjusts the video tempo to maintain video synchronization, but in an alternate embodiment the method adjusts the audio tempo to maintain synchronization. Once a video tempo value has been determined, the preferred method adjusts this video tempo value by applying a smoothing function, i.e., a weighted average of prior tempo values, to the determined tempo value. If the synchronization error is determined to be greater than a defined tolerance, i.e., if the audio and video data streams are more than a certain number of flames out of

sync, and if the tempo value is not equal to the last tempo value previously sent to the video driver, then the method adjusts the video frame speed by passing the tempo value to the video driver.

If the synchronization error is approximately 0, i.e., the audio and video data streams are substantially in sync, and the prior determined tempo value passed to the video driver was not the nominal rate, i.e., the rate intended to exactly match the audio rate, the method passes a video tempo value at the nominal rate to the video driver. In other words, if the audio and video data streams are in sync, a tempo value at the nominal rate is passed to the video driver. This removes any affects of the smoothing function, which otherwise would change the tempo value to other than the nominal rate.

The method also determines if the audio is too far ahead of the video and if the audio is playing. If so, the audio is paused to allow the video to catch up. If the method determines that the audio is paused and that the video has caught up, the method restarts the audio. The method saves the video tempo value for comparison during the next call by the periodic timer and surrenders control to the operating system until called again.

Therefore, the present invention provides an improved method of synchronizing the audio and video data streams during a multimedia presentation to provide a correctly synchronized presentation. The present invention permits the use of existing software drivers and multimedia operating systems. Further, the method of the present invention operates independently of where the audio and video data are stored as well as the type of operating system or drivers being used. Thus the present invention operates regardless of whether the audio and video data are interleaved in one file, stored on different media, or stored in separate computer systems. Also, the present invention does not require any type of time stamping or tagging of data, and thus does not require any modification of the video or audio data. Further, the present invention operates regardless of the type of compression/decompression algorithm used on the video data.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings, in which:

FIG. **1** illustrates a block diagram of a multimedia computer system according to one embodiment of the invention;

FIG. **2** is a block diagram illustrating a typical multimedia control system;

FIG. **3** is a block diagram illustrating a prior art multimedia software architecture;

FIG. **4** is a block diagram illustrating a multimedia software architecture incorporating the synchronization method of the present invention; and

FIGS. **5A–B** are flowchart diagrams illustrating operation of the synchronization method of the present invention.

FIG. **6** illustrates audio and video data streams having a common starting point.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Multimedia Computer System

Refering now to FIG. **1**, a block diagram illustrating a multimedia computer system according to one embodiment of the present invention is shown. It is noted that FIG. **1** illustrates only portions of a functioning computer system, and those elements not necessary to the understanding of the operation of the present invention have been omitted for simplicity. As shown, the multimedia computer system includes a CPU **102** coupled to a host bus **106**. Main memory **104** is also coupled to the host bus **106**. The host bus **106** is coupled to an expansion bus **112** by means of a bus controller **110**. The expansion bus may be any of various types including the AT (advanced technology) bus or industry standard architecture (ISA) bus, the EISA (extended industry standard architecture) bus, a microchannel (MCA) bus, etc. A video card or video adapter such as a VGA (video graphics array) card **120** is coupled to the expansion bus **112** and is adapted to interface to a video monitor **122**, as shown. The computer system may also include a video accelerator card **124** for performing compression/decompression (codec) functions. However, in the preferred embodiment the computer system does not include a video accelerator card. An audio card or sound card **130** is also coupled to the expansion bus **112** and interfaces to a speaker **132**. The audio board **130** is preferentially a Sound Blaster II brand card made by Creative Labs, Inc. of Milpitas, Calif.

Various mass storage devices are also coupled to the expansion bus **112**, preferably including a CD-ROM **140**, and a hard drive **142**, as well as others. One or more of these mass storage devices store video and audio data which is used during presentation of a multimedia display. The audio and video data may be stored in any of a number of formats and may be stored on different media. Further, the audio and video data may be stored on media located in other computer systems that are connected to the computer system via a network. Thus the present invention can operate in a distributed environment.

It is noted that a multimedia computer system according to the present invention may be configured in any of a number of ways. For example, the video and audio card **120** and **130**, as well as one or more of the mass storage devices **140** or **142** may be coupled to a CPU local bus such as the PCI (peripheral compact interconnect) bus, or the VESA (Video Enhanced Standards Association) local bus, as desired. Various other computer configurations are also contemplated, such as a distributed system.

In the preferred embodiment of the invention, the multimedia computer system illustrated in FIG. **1** operates using the Windows 3.1 Operating System from Microsoft Corporation of Redmond, Washington. The computer system also preferably includes the Microsoft Windows multimedia extension software, including Microsoft's Media Control Interface and associated drivers. The Windows Media Control Interface (MCI) is a set of high-level commands that provide a device-independent interface for controlling multimedia devices and media files. The MCI command set is designed to provide a generic core set of commands to control different types of media devices. Because of the high level of device independence provided by the MCI command set, a programmer can use MCI commands rather than a low level API to access the multimedia capabilities of Windows. It is noted that the computer system may use other operating systems and other multimedia software, as desired.

The computer system includes video and audio drivers which interface to video and audio hardware, respectively. The audio driver interfaces between the multimedia operating system and the audio card **130**. The video driver interfaces between the operating system and the video accelerator card, if any. In the preferred embodiment, which does not

include a video accelerator **124**, the video driver does not actually interface to any video hardware, but rather performs various video data processing on the main CPU **102**. In the preferred embodiment, the video driver is comprised in the Intel Audio-Visual Kernel (AVK). The audio driver is preferably an MCI compliant WAV driver corresponding to the respective audio card **130**.

The computer system also includes a synchronization method according to the present invention which synchronizes the audio and video data streams during a multimedia presentation to ensure that the appropriate sounds are generated by the speaker **132** when the corresponding images are being displayed by the video monitor **122**.

Multimedia Software Architecture of the Preferred Embodiment

Referring now to FIG. **2**, the Microsoft Windows' Multimedia Extensions Software Architecture is illustrated. As shown in FIG. **2**, a multimedia application **200** directs a computer system to present a multimedia display by interfacing to the hardware through the operating system and various device driver layers. The block **210** includes the Windows Kernel and Graphics Device Interface (GDI), i.e., the bulk of the Windows operating system. As shown, the multimedia application **200** interfaces through the Windows operating system **210** to Windows device drivers **220**. The device drivers **220** interface to the various elements in the computer system including the printer, hard drive **142**, video monitor **122**, etc.

The block **240** comprises the Windows Multimedia Extensions Software. This translation layer isolates applications from device drivers and centralizes device-independent code. The translation layer **240** translates a multimedia function call into a set of Media Control Interface ("MCI") calls which interface to Media Control Interface drivers **250**. As shown, the Media Control Interface drivers interface to mass storage devices **260** and **270** such as the CD-ROM **140** or hard drive **142**. The Media Control Interface layer also communicates with MCI compliant multimedia device drivers **230** using a set of low level functions, as shown. The multimedia hardware device drivers **230** directly control a multimedia device such as an audio card **130** or video accelerator **124**. For more information on the Media Control Interface layer **240**, please see *Discover Windows* 3.1 *Multimedia* by Roger Jennings (Que Corp. 1992) chapters 23 and 24, which is hereby incorporated by reference. Please also see generally the *Microsoft Multimedia Programmer's Reference* and the *Microsoft Windows Multimedia Programmer's Workbook*, available from Microsoft Corporation, which are both hereby incorporated by reference.

The recording and presentation of multimedia displays is handled by the Windows Multimedia Extension Software in conjunction with the individual MCI drivers. Currently, not all multimedia drivers support MCI commands and command options. In particular, optional commands used by some devices are not supported. An example of an option command is "set," which sets the operating state of a device. Such a command would support the option "tempo 200," for example, as a means of controlling the speed of the playback device. Another example of an option command is the "pause" command, which suspends operation of a playback device, but leaves the device ready to resume playing immediately.

One embodiment of the present invention avoids use of MCI commands for querying the audio and video drivers and controlling the tempo and pausing of playback devices. In this embodiment the synchronization method of the present invention makes calls directly to the API of the audio and video drivers to obtain the necessary information and control the tempo of one of the respective data streams to maintain synchronization. The preferred embodiment of the invention uses the MCI interface to access the respective multimedia device drivers to query the drivers as well as adjust the tempo of the data streams. In alterative embodiments of the invention, the synchronization method uses MCI commands to interface to the audio driver and uses the direct API to access the video driver, or vice versa.

Multimedia storage devices can be classified as either simple devices or compound devices. Simple devices do not require a data file for playback, and videodisc players and CD audio players are examples of simple devices. Compound devices require a data file for playback and examples of compound devices include digital video players and waveform audio players. The preferred embodiment of the invention is used to synchronize audio and video data streams from compound devices.

Prior Art Multimedia System

Referring now to FIG. **3**, a prior art multimedia system that does not include the synchronizing method of the present invention is shown. As shown in FIG. **3**, audio and video data are stored in a multimedia system using one of a variety of storage devices **301**, **302**, and **303**, including the hard disk **142** or CD-ROM **140**. If the audio and video data are stored in the audio-video interleave (AVI) file format, then the audio and video data are interleaved together on the storage media in the same file. Alternatively, the audio and video data are stored on different storage media, perhaps on different computer systems connected via a network. As previously noted, the present invention operates regardless of whether the audio and video data are interleaved together, stored on separate media, or stored on separate computer systems.

The audio and video data are provided to the CPU **102**, which is preferably executing the Microsoft Windows Operating System **307**, as well as the Microsoft Multimedia Extension software. The Multimedia Extension software invokes the media control interface (MCI) layer software **309**, which in turn invokes respective MCI digital drivers **311** to provide the respective data streams to the respective hardware subsystems. As shown, the MCI digital driver **311** communicates with the video driver in the Intel audio-video kernel (AVK) **341**. The video driver in the AVK performs various processing on the video data and interfaces to video accelerator hardware **124**, if any. The video driver in the AVK **341** also monitors the frame number of the video frame being played. The video data is then provided to the video frame buffer **345** in the video adaptor (VGA card) **120** where it is then displayed on the video monitor **122**. The MCI digital driver **311** also communicates with the audio driver **331** to provide the audio data **313** to the respective hardware audio card **130** to the speaker **132**. The audio driver **331** is preferably an MCI-compliant driver. Thus, audio data is read from the respective storage device and provided to the audio card **130** by the MCI driver **311** and the audio drivers **331** executing on the CPU **102**. The audio driver **331** monitors the number of bytes of audio data processed from the start of a particular set of data.

As discussed in the background section, prior art multimedia systems provide generally unsynchronized audio and video outputs during a multimedia presentation due to the

inherent difficulty of synchronizing separate audio and video data streams passing through separate audio and video subsystems and controlled by separate audio and video drivers. Numerous factors can affect the playback of the audio and video data streams, including the greater amount and more variable amount of processing required for the video data as well as other demands that can be made on the system.

As discussed above, a large amount of processing may be required before the video data can be displayed on the video monitor **122**. For example, if the video's color depth is higher than the display's, as when an AVI file is played with 16-bit video on an 8-bit display, colors must be dithered to fit within the display's color restrictions. Also, if the playback window size differs from the resolution at which the video was captured, each frame must be scaled. Video and audio data processing can be adversely affected by a number of other factors, including a slow hard disk, a slow CD-ROM controller, and a slow display controller or audio card. During this process, it becomes virtually impossible for the environment to maintain the audio and video data properly synchronized while managing other critical tasks. As discussed in the background section, prior art methods, to the extent there are any, have proved inadequate in maintaining synchronization between video and audio data streams.

Multimedia Software Architecture Including the Preferred Embodiment of the Present Invention

Referring now to FIG. **4**, operation of the preferred embodiment of the present invention is illustrated. Logical blocks in FIG. **4** that are similar to those shown in FIG. **3** are designated with the same reference numeral for convenience. As discussed above, the preferred embodiment uses an MCI compatible interface to perform the synchronization method of the present invention. The preferred embodiment also uses the video driver in the Intel Audio Video Kernel (AVK) **341**. Because of the high level of device independence provided by the MCI interface, the multimedia capabilities of Microsoft Windows can be accessed through MCI protocols, rather than through low-level Application Program Interfaces (API). The preferred embodiment uses the MCI protocol and commands to interface with the video and audio drivers. These protocols are found in the *Microsoft Windows Software Development Kit, Multimedia Programmer's Guide*, Document Number PC30253-0492 which can be obtained from Microsoft Corporation of Redmond, Washington. (facsimile number 206-936-7329). In an alternative embodiment of the present invention, as mentioned above, the method of the present invention avoids the MCI layer and instead communicates directly with the API of the video and audio drivers. Bypassing the MCI interface layer results in a slight speed increase due to the decreased overhead. The source code listing following this description implements an embodiment of the invention that bypasses the MCI layer and instead communicates directly with the API of the audio and video drivers.

As shown in FIG. **4**, video data and commands pass from the MCI digital driver **311** to the video driver in the AVK **341** via the data path **315**. The AVK in turn provides data to the video hardware **124**, if any, which in turn provides the video data to the video frame buffer **345** in the video adaptor **120**.

Audio data and commands are transferred from the MCI digital driver **311** to the audio driver **331** via data path **313**. The audio driver **331** in turn provides the data to the audio card **130** via data path **333**, and the speaker **132** produces sound corresponding to the audio data.

The synchronization method of the present invention is performed by synchronization module block **421**. The synchronization module **421** comprises a method that is preferably implemented in software, and a source code listing of one embodiment of this method is located at the end of this specification. As noted above, the source code listing at the end of this specification operates by interfacing directly to the API of the audio and video drivers rather than going through the MCI interface layer. Otherwise the source code listing is similar to the preferred method. The MCI digital driver **311** provides a signal to the synchronization module **421** over path **417**. The computer system includes a timer which periodically interrupts the MCI layer **309** and MCI driver **311** and directs the MCI layer **309** to invoke the synchronization module **421** of the present invention. When the synchronization module **421** is invoked, the synchronization method queries the audio and video drivers **331** and **341** for the current position of the audio and video data. The synchronization module **421** is shown connected to the AVK video driver **341** and the audio driver **331**. As noted above, the synchronization module **421** of the preferred embodiment of the invention interfaces to the AVK video driver **341** and the audio driver **331** through the MCI interface layer. In contrast, the source code listing at the end of this specification implements an embodiment that accesses the AVK video driver **341** and audio driver **331** directly via the API of the respective drivers.

The audio driver **331** provides audio position information to the synchronization module **421** over path **429**. The AVK video driver **341** provides video frame position data over signal path **423** to the synchronization module **421**. The synchronization module **421** uses the audio and video frame rate information to compute a video tempo value that is provided to the AVK video driver **341**. Video tempo and pause commands are conveyed from the synchronization module to the AVK **341** over signal path **425**, preferably routed through the MCI layer **309** as discussed above. Also, in the preferred embodiment, the synchronization module **421** provides a pause command to the audio driver **331**. In an alternate embodiment, the present invention maintains synchronization by adjusting the audio tempo, and the synchronization module **421** generates an audio playback tempo command that is conveyed to the audio driver **331** over signal path **427**.

Synchronization Method—Flowchart

Referring now to FIGS. **5A** and **5B**, a flowchart diagram illustrating operation of the synchronization method performed by the synchronization module **421** according to the present invention is shown. The main portion of the synchronization method is located at lines **151–327** of the source code listing at the end of this specification. The synchronization method makes a function call to a function referred to as audframe_decupl_aud_dev, which computes the audio frame number from the audio position. This function is located generally at lines **1–150** of the source code listing.

When the synchronization method is invoked, in step **502** the method determines if this is the first time the method has been invoked. If so, then in step **504** the method calls the audio driver to obtain the wave rate, i.e., how many kilohertz at which the audio is operating. In step **506** the method calculates and stores the number of bytes that are in an audio frame that is equivalent to a corresponding video frame. The method obtains the wave rate at which the audio is playing, determines how many bytes of audio are played each second, then calculates the equivalent number of bytes for

each video frame using the known video frame rate. The equation used in this calculation is:

```
bytes_per_frm_x100000=Avio→AudStrms[0]
    .SamplesPerSecond*Avio→AudStrms[0].FrameRate/10 UL;
```

In step **508** the synchronization method initializes other variables. For example, the method initializes a previous tempo variable to a starting value, preferably a nominal value. As discussed further below, this previous tempo variable is used to record the prior tempo variable provided to the video driver the last time the synchronization module was invoked. Other variables can be initialized as desired. If the synchronization module is not being called for the first time in step **502**, then operation proceeds directly to step **510**.

It is noted that the preferred embodiment of the invention operates as shown in FIG. 5A in steps **502–508**. However, in an alternate embodiment, steps **502–508** are performed elsewhere, such as in the audio driver **331** or the MCI layer **309**. It is also noted that steps **502–508** are not included in the source code listing at the end of this specification.

In step **510** the method determines the current video frame number. The synchronization method of the present invention calls the respective video driver **341** controlling the video hardware (if any) to determine what video frame number is currently being played. This call preferably uses MCI interface commands. In step **512** the method calls the respective audio driver to determine the current audio position, i.e., which audio byte is currently being played. This call also preferably utilizes MCI commands. In an alternate embodiment, the call to the video driver is made directly to the API of the video driver, and the call to the audio driver involves a call directly to the API of the .WAV audio driver **331** to determine what audio byte is currently being played. In step **514** the method then calculates the equivalent audio frame number being played using the audio frame rate value calculated and stored in step **506**. As shown in the source code listing at the end of this specification, step **514** invokes a function referred to as audframe_decupl_aud_dev. This function calculates the current audio frame number using a fraction representing the number of bytes per equivalent audio frame to determine the audio frame number. This function preferably does not use floating point numbers in the calculation due to the perceived unreliability of floating point numbers in some programming environments.

Referring to FIGS. **5A** and **5B** and **6**, the preferred embodiment of the present invention requires that there be a common starting point for the audio and video data. FIG. **6** illustrates an audio data stream **604** and a video data stream **606** having a common starting point **602**. Each audio data stream **604** includes audio frames **608** having audio data, and each video data stream **606** includes video frames **610** having video data. There is a time index of zero where the audio data stream **604** and video data stream **606** are both in synchrony, such that a first byte of the audio data stream **604** and a first byte of the video data stream **606** are generated simultaneously.

In step **516** the method then calculates the synchronization error quantity. Clearly, because audio data stream **604** and video data stream **606** have a common starting point **602**, calculating the synchronization error quantity essentially involves subtracting the current video frame number from the current audio frame number to determine the number of frames by which the audio and video are out of sync.

In step **518** the method determines if the audio is too far ahead of the video and if the audio is still playing. In the preferred embodiment the method determines if the audio is more than 5 frames ahead of the video in step **518**. If the audio is determined to be too far ahead and is also playing in step **518**, then the method stops the audio in step **520** and then advances to step **522**. If the audio is either not too far ahead, i.e., not more than 5 frames ahead, or the audio is not playing, then operation advances directly to step **522**. It is noted that if the audio is determined to be too far ahead in step **518**, i.e. more than 5 frames ahead of the video, then the synchronization method of the present invention may not be working, i.e., the video tempo is not being set properly. Another possibility is that the synchronization method is not being called often enough. It is noted that if the video advances too far ahead of the audio, then the video is simply slowed down using a lower video tempo.

In step **522** the method determines if the audio is paused and the video has caught up to the audio. In the preferred embodiment, the video is considered to have caught up to the audio if the audio is less than 2 frames ahead of the video. If the audio is paused and the video is determined to have caught up to the audio, then the audio is restarted in step **524**. Operation then advances to step **526** (FIG. **5B**). If either the audio is not paused or the video has not caught up to the audio, then operation proceeds directly to step **526**.

In step **526** the method selects a synchronization adjustment factor, referred to as a tempo value, for the video driver using a lookup table. In the source code listing at the end of this specification, the video tempo value is actually selected from a case statement. As noted above, in the preferred embodiment the synchronization method adjusts the video frame rate or video tempo to maintain the audio and video data streams in sync. However, it is noted that in the present invention either the audio or video stream rates can be adjusted as desired. For example, the method could slow down or speed up the video frame rate or slow down or speed up the audio frame rate as desired to maintain the respective audio and video data streams in sync. It is noted that adjusting the audio data stream may be a simpler procedure than adjusting the video data stream. The audio data stream was not adjusted in the preferred embodiment because of concerns that the user might be able to hear the audio adjustments. However, experimentation has shown that adjustments to the audio data stream would generally not be detectable by the user if the synchronization method of the present invention was invoked a sufficient number of times each second.

In step **528** the method determines if the video has started to play. If not, then the method sets the video tempo value to a slow value. This is done merely to begin the video portion of the presentation at a slow rate. If the video has started to play, operation proceeds directly to step **532**. In step **532** the method adjusts the video tempo value using a smoothing or dampening function. The preferred embodiment uses a smoothing formula which combines one half of the current tempo value plus one half of the previous tempo value. This smoothing function operates to prevent overcompensation and to add stability to the synchronization method, thus allowing for smoother synchronization. The adjustment performed in step **532** is similar to a damping function.

In step **534** the method determines if the audio is paused. If the audio is determined to not be paused in step **534**, then in step **536** the method determines if audio data is available. If audio data is determined to not be available in step **536**, then it is assumed that the multimedia presentation does not

include any audio data. In this case the method exits since it not necessary to adjust the video playback speed or tempo if there is no audio component. If audio data is determined to be available in step **536**, then operation advances to step **542**.

If the audio is determined to be paused in step **534**, then in step **538** the method determines if the audio is ahead of the video or if the audio status is reported as bad. If the audio is determined to be equal to or behind the video in step **538**, then operation advances to step **542**. If the audio is determined to be ahead of the video in step **538** or if the audio status is reported as bad, then the method sets the video tempo to a nominal rate, i.e. , a rate which is calculated to be equal to the average speed of the audio playback. The video tempo is set to the nominal rate in step **540** because it is not necessary to set the video rate to large (fast) tempos if the audio is paused and the audio is ahead of the video. In cases where the audio is paused and is ahead of the video, this typically means that the user has either clicked on the PAUSE or STEP button during the presentation, or that the audio has been halted because the audio and video were too far out of sync.

In step **542** the method determines if the synchronization error calculated in step **516** is greater than a set tolerance. In the preferred embodiment the tolerance is set to 1 frame. Thus the synchronization method does not adjust the video tempo unless the audio and video are at least a certain amount out of sync. If the audio and video are in sync or are relatively close to being in sync, then the tempo is not adjusted. This avoids having to call the video driver to adjust the tempo and thus reduces the overhead caused by synchronization. If the synchronization error is greater than the tolerance in step **542**, then in step **543** the method determines if the tempo value is equivalent to the last_tempo value, i.e., the tempo value sent to the video driver the last time the synchronization method was executed. If the current tempo value equals the last_tempo value, then operation advances to step **554**. In this instance the video driver is already operating at this tempo, and thus there is no need to call the video driver to set the tempo value to the same value. This also serves to reduce the overhead of the synchronization method. If the tempo value does not equal the last_tempo value in step **543**, then in step **544** the method adjusts the video frame rate using the tempo value determined in step **526** and adjusted in step **532**. As noted above the method preferably adjusts the video frame rate using MCI interface calls. In an alternate embodiment, as shown in the source code listing, the synchronization method passes the video frame rate or tempo value directly to the video driver, using the AvkGrpTempo function call. The video driver uses the received number to adjust the video frame rate. Operation then advances to step **554**.

If the synchronization error is determined to not be greater than the tolerance in step **542**, then in step **546** the method determines if the synchronization error is 0 and if the previous tempo was not set to the nominal rate. If either the synchronization error is not 0 or the previous tempo was set to the nominal rate, then operation advances to step **554**. In this instance it is not necessary to adjust the tempo because the synchronization error was determined to be less than the tolerance in step **542**. If the synchronization error is **0** and the last tempo is not equal to the nominal rate, then an extra stabilizing effect is added. This extra stabilizing effect is referred to as a "lock-down" function. In step **548** the method sets the tempo to the nominal rate and in step **550** the method adjusts the video frame rate by making the appropriate MCI interface call to the video driver. In step **552** the method sets the last_tempo variable to the nominal rate to prevent steps **548–552** from being performed the next time the synchronization method is called.

As noted above, steps **548–552** are performed when the synchronization error is **0** but the prior tempo was not set to the nominal rate. Here it is desirable to eliminate the effects of the smoothing function applied in step **532**. If the audio and video data streams are in sync, the tempo value produced by the smoothing function in step **532** will suggest that the two streams are out of sync because part of the tempo value calculation is the weighted average of the previous tempo value with the current value. Thus, if the current tempo value is zero, but a previous adjustment was required, the synchronization method would report that a tempo adjustment is necessary. Therefore, in this instance the tempo is set to the nominal rate just as if the smoothing function had not been applied.

Thus, this lock-down function ensures that the synchronization method does not overcompensate when the audio and video are in sync. In other words, this function adds stability by locking on a point where the audio and video are in sync to prevent overcompensation from occurring, i.e., primarily to prevent the smoothing function applied in step **532** from pushing the audio and video out of sync. Without this lock-down function, if the audio and video data streams were in sync, the smoothing function would cause the streams to fall out of sync, i.e., would cause the video data stream to oscillate between being ahead of or behind the audio stream.

In step **554** the method saves the tempo value for the next call by the periodic timer. The synchronization method then completes.

Conclusion

Therefore, a method and apparatus for synchronizing the audio and video portions of a multimedia display is shown. This method provides superior synchronization over methods found in the prior art.

Although the method and apparatus of the present invention has been described in connection with the preferred embodiment, it is not intended to be limited to the specific form set forth herein, but on the contrary, it is intended to cover such alternatives, modifications, and equivalents, as can be reasonably included within the spirit and scope of the invention as defined by the appended claims.

We claim:

1. A method for synchronizing audio and video data streams having a common starting point during a multimedia presentation, comprising the steps of:

determining a current position of the video data stream relative to the common starting point;

determining a current position of the audio data stream relative to the common starting point;

calculating a synchronization error related to a difference between the respective video data stream and audio data stream current positions using the current positions of the audio and video data streams;

adjusting a tempo of one of the data streams based on the synchronization error if necessary to place the audio and video data streams in synchrony;

repeating the determining steps and the steps of calculating and adjusting during the multimedia presentation to maintain the audio and video data streams in synchrony.

2. The method of claim **1**, further comprising:

determining if the synchronization error is greater than a tolerance value after the step of calculating and prior to the step of adjusting, and

wherein the step of adjusting the tempo is performed only if the synchronization error is greater than the tolerance value.

3. The method of claim 1, further comprising:

determining a tempo value from the synchronization error calculated in the step of calculating prior to the step of adjusting; and

wherein the step of adjusting comprises adjusting the tempo of one of the data streams using the determined tempo value.

4. The method of claim 3, further comprising:

storing the determined tempo value;

calculating a second synchronization error by repeating at a subsequent time the steps of determining a current position of the video data stream, determining a current position of the audio data stream, and calculating a synchronization error;

determining a second tempo value from the second synchronization error; and

repeating the step of adjusting based on the second synchronization error if the second determined tempo value does not equal the stored determined tempo value.

5. The method of claim 4, further comprising:

applying a smoothing function to the second determined tempo value using the stored determined tempo value and second determined tempo value prior to the step of adjusting, wherein the smoothing function prevents overcompensating the tempo adjustment and adds stability to the synchronizing method.

6. The method of claim 5, further comprising:

determining if the synchronization error is 0 and if the immediately prior tempo value was not equal to a nominal rate prior to the step of adjusting; and

wherein the step of adjusting the tempo comprises setting the tempo to the nominal rate if the synchronization error is 0 and the immediately prior tempo value was not equal to a nominal rate.

7. The method of claim 1, wherein the step of adjusting the tempo comprises adjusting the tempo of the video data stream.

8. The method of claim 1, wherein the step of adjusting the tempo comprises adjusting the tempo of the audio data stream.

9. The method of claim 1, further comprising:

determining if the audio data stream is far ahead of the video data stream and if the audio is playing after the steps of determining respective current positions; and

halting playback of the audio data steam if the audio data steam is far ahead of the video data steam and the audio is playing.

10. The method of claim 9, further comprising:

determining if the audio playback is paused and if the video data stream has approximately caught up to the audio data stream; and

restarting the audio playback if the audio playback is paused and the video data stream has approximately caught up to the audio data stream.

11. The method of claim 1, wherein the video data stream is processed in a video subsystem and the audio data stream is processed in an audio subsystem such that the video data stream is decoupled from the audio data stream.

12. The method of claim 13, wherein each video frame has a video frame number relative to the common starting point, wherein the step of determining the current position of the video data stream comprises determining a current video frame number being played.

13. The method of claim 1, wherein at any point in time during the multimedia presentation an audio frame and a video frame are being played and the step of determining the current position of the audio data stream includes the steps of:

determining which audio byte is currently being played;

obtaining a frequency at which the audio is being played;

determining how many bytes of audio are being played per unit of time;

determining a number of bytes per audio frame equivalent to a number of bytes per video frame: and

determining an equivalent audio frame number being played relative to the common starting point.

14. The method of claim 12, wherein the step of calculating a synchronization error comprises calculating a difference between the current video frame number and the determined equivalent audio frame number.

15. The method of claim 1, wherein the synchronizing method is periodically performed each time after the expiration of a predetermined time interval.

16. A computer system which synchronizes audio and video data streams, having respective audio and video data, during an audio-visual display, comprising: one or more storage devices for storing the audio and video data for the audio-visual display;

a video monitor coupled to the one or more storage devices for generating a video display corresponding to the video data in the video data stream;

a speaker coupled to one or more storage devices for generating sounds corresponding to the audio data in the audio data stream;

one or more data paths for transmitting the audio and video data streams corresponding to the audio and video data from the one or more storage devices to the speaker and the video monitor, respectively;

means coupled to the one or more storage devices, the one or more data paths, the video monitor and the speaker for obtaining a current position of the video data stream relative to a common starting point of the audio and video data streams;

means coupled to the one or more storage devices, the one or more data paths, the video monitor and the speaker for obtaining a current position of the audio data stream relative to the common starting point;

means coupled to both the means for obtaining for calculating a synchronization error related to a difference between the respective video data stream and audio data stream current positions using the current positions of the audio and video data streams; and

means coupled to the one or more data paths and the synchronization error calculating means for adjusting a tempo of one of the data streams based on the synchronization error if necessary to place the audio and video data streams in synchrony; and

wherein the means for obtaining, means for calculating, and means for adjusting repeat their respective functions during the audio-visual display to maintain the audio and video data streams in synchrony.

17. The computer system of claim 16, further comprising:

means coupled to the calculating means for determining if the synchronization error is greater than a tolerance value;

wherein the means for adjusting the tempo adjusts the tempo only if the synchronization error is greater than the tolerance value.

18. The computer system of claim 16, further comprising:

means for determining a tempo value from the synchronization error calculated by the means for calculating;

wherein the means for adjusting adjusts the tempo of one of the data streams using the determined tempo value.

19. The computer system of claim 18, further comprising:

means coupled to the tempo value determining means for storing the determined tempo value, wherein the stored determined tempo value is used as a prior tempo value the next time the synchronizing method is performed;

means coupled to the tempo value determining means and the storing means for determining if the determined tempo value equals the prior tempo value;

wherein the means for adjusting is not executed if the determined tempo value equals the prior tempo value.

20. The computer system of claim 18, further comprising:

means coupled to the tempo value determining means for storing the determined tempo value;

means coupled to the tempo value determining means and the storing means for applying a smoothing function to the determined tempo value using a tempo value stored by the means for storing, wherein the smoothing function prevents the means for adjusting the tempo from overcompensating the tempo adjustment and adds stability to the tempo adjustment.

21. The computer system of claim 20, further comprising:

means coupled to the calculating means for determining if the synchronization error is 0 and if the immediately prior tempo value was not equal to a nominal rate; and

wherein the means for adjusting the tempo comprises means for setting the tempo to the nominal rate if the synchronization error is 0 and the immediately prior tempo value was not equal to a nominal rate.

22. The computer system of claim 16, wherein the means for adjusting the tempo adjusts the tempo of the video data stream.

23. The computer system of claim 16, wherein the means for adjusting the tempo adjusts the tempo of the audio data stream.

24. The computer system of claim 16, further comprising:

means coupled to the one or more data paths for determining if the audio data stream is far ahead of the video data stream and if the audio is playing; and

means for halting playback of the audio data stream is the audio data stream is far ahead of the video data stream and the audio is playing.

25. The computer system of claim 24, further comprising:

means coupled to one or more data paths for determining if the audio playback is paused and if the video data stream has approximately caught up to the audio data stream;

means coupled to the one or more data paths for restarting the audio playback if the audio playback is paused and the video data stream has approximately caught up to the audio data stream.

26. The computer system of claim 16, wherein the one or more data paths comprises an audio data path for transmitting the audio data stream and a video data path for transmitting the video data stream.

27. The computer system of claim 28, wherein the means for obtaining the current position of the video data stream comprises means for obtaining a current video frame number.

28. The computer system of claim 16, wherein the audio data stream and the video data stream include audio frames and video frames, respectively, each audio and video frame includes a respective frame number relative to the common starting point, the means for obtaining the current position of the audio data stream further comprising:

a means for determining which audio byte is currently being played;

wherein said means for obtaining the current position of the audio data stream includes means for calculating an equivalent audio frame number relative to the common starting point using the audio byte currently being played, an audio play frequency, using a number of bytes of audio played per unit of time, and using a number of bytes per audio frame equivalent to a number of bytes per video frame.

29. The method of claim 27, wherein the means for calculating a synchronization error calculates a difference between the current video frame number and the calculated equivalent audio frame number.

30. The method as in claim 1 wherein the common starting point is a time index of zero where the audio and video data streams are both in synchrony, such that a first byte of audio data and a first byte of video data are generated simultaneously.

31. The method as in claim 1 wherein the multimedia presentation utilizes a computer system having a video driver, and wherein the video data stream includes video frames, and wherein the video data stream current position determining step comprises querying the video driver to determine a current video frame number being played, the method further comprising:

determining a tempo value using the synchronization error;

wherein the tempo adjusting step comprises passing the tempo value to the video driver.

32. The method as in claim 1 wherein the multimedia presentation utilizes a computer system having an audio driver and the audio data stream current position determining step comprises the step of:

querying the audio driver to determine the current audio data stream position.

33. The method as in claim 1 wherein the video data stream and the audio data stream are respectively free from time stamps, and the video data stream is free from any timing information relative to the audio and video data stream correspondence.

34. The method as in claim 1 wherein the tempo is a rate of one of the data streams.

35. The method as in claim 5 wherein the smoothing function combines one half of a current tempo value plus one half of the prior tempo value.

36. The computer system as in claim 16 wherein the current position of the video data stream is represented by a video frame number, the computer system further comprising:

a video driver means, coupled to the means for obtaining a current position of the video data stream, for passing the video frame number to the means for obtaining a current position of the video data stream.

37. The computer system as in claim 16 wherein the current position of the audio data stream is represented by an audio frame number, the computer system further comprising:

an audio driver means, coupled to the means for obtaining a current position of the audio data stream, for passing the audio frame number to the means for obtaining a current position of the audio data stream.

38. The computer system as in claim 16 wherein the common starting point is a time at which a first byte of the audio data and a first byte of the video data are in synchrony.

39. The computer system as in claim 16 wherein the tempo is a rate of one of the data streams.

40. The computer system as in claim 20 wherein the smoothing function combines one half of a current tempo value plus one half of the prior tempo value.

* * * * *

US005719786A

United States Patent [19]

Nelson et al.

[11] Patent Number: 5,719,786

[45] Date of Patent: Feb. 17, 1998

[54] **DIGITAL MEDIA DATA STREAM NETWORK MANAGEMENT SYSTEM**

[75] Inventors: **David L. Nelson**, Framingham; **Premkumar Uppaluru**, North Andover; **Pasquale Romano**, Boston; **Jeffrey L. Kleiman**, Lexington, all of Mass.

[73] Assignee: **Novell, Inc.**, Provo, Utah

[21] Appl. No.: **13,009**

[22] Filed: **Feb. 3, 1993**

[51] **Int. Cl.**[6] **G06K 15/00**

[52] **U.S. Cl.** ..... **364/514 A**; 348/19

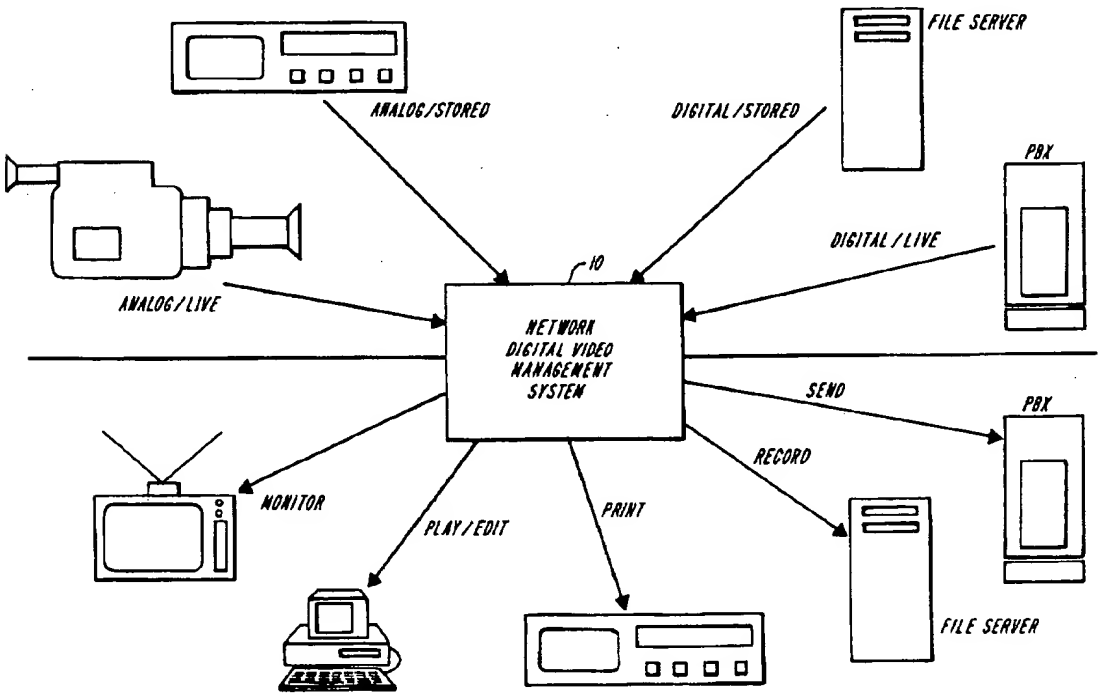
[58] **Field of Search** ..... 364/514 A, 715.02; 348/19, 24; 395/114, 154, 118

Primary Examiner—Ellis B. Ramirez
Assistant Examiner—Thomas Peeso
Attorney, Agent, or Firm—Snell & Wilmer

[57] **ABSTRACT**

A computer-based media data processor for controlling transmission of digitized media data in a packet switching network. When the processor receives a request from a network client node for presentation of specified media data stream presentation unit sequences the processor in response retrieves media data from a corresponding media access location, determines the media data type of each presentation unit in the retrieved media data, and designates each retrieved presentation unit to a specific media data presentation unit sequence based on the media data type determination for that presentation unit. The processor then assembles a sequence of presentation descriptors for each of the specific presentation unit sequences, all presentation descriptors in an assembled sequence being of a common media data type, and then assembles transmission presentation unit packets each composed of at least a portion of a presentation descriptor and its media data, all presentation descriptors and media data in an assembled packet being of a common media data type. The assembled packets are then released for transmission via the network to the client processing node requesting presentation of the specified presentation unit sequences.

105 Claims, 12 Drawing Sheets



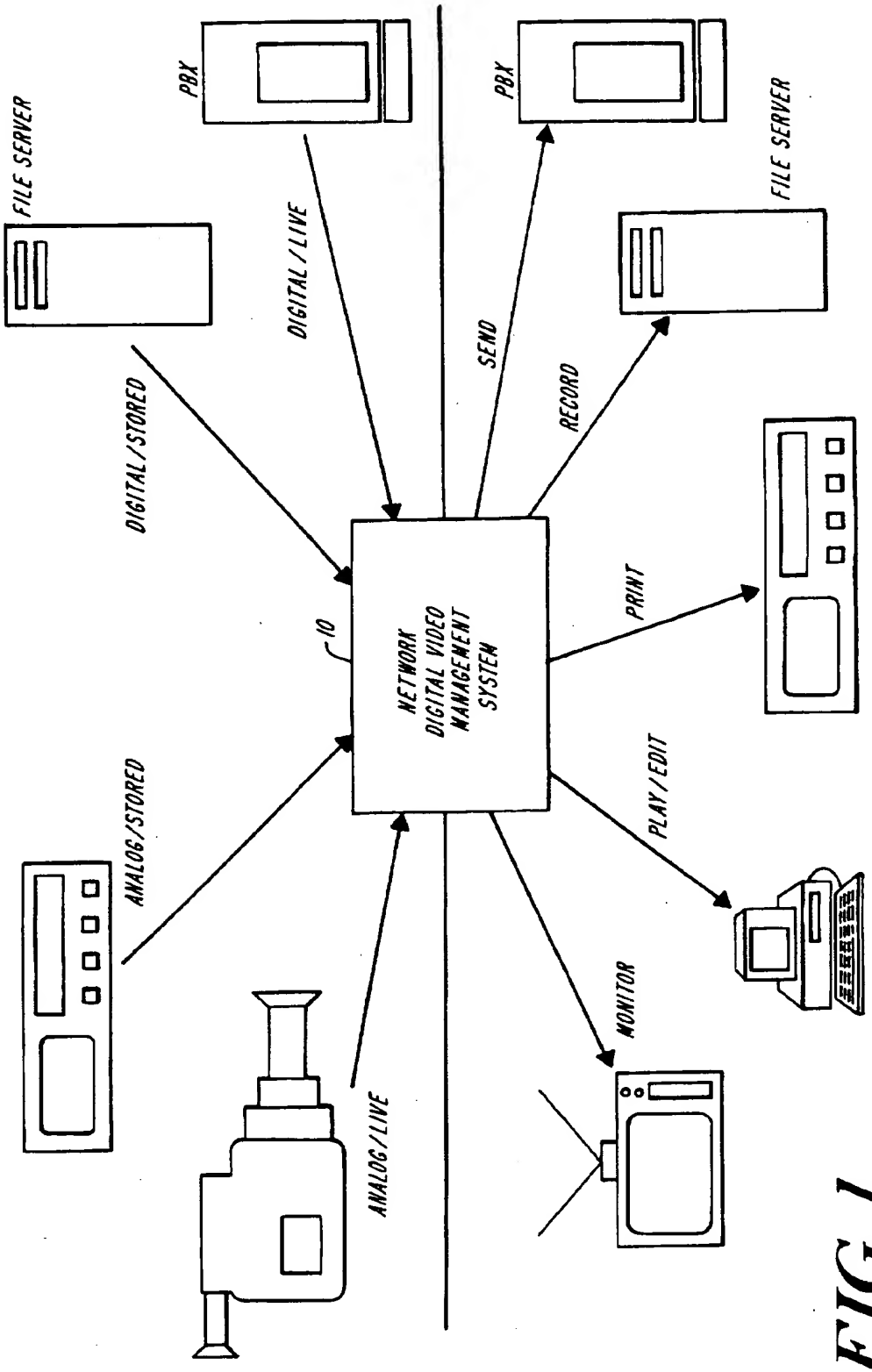

PBX
PBX
FILE SERVER
FILE SERVER
DIGITAL/LIVE
SEND
RECORD
DIGITAL/STORED
10
NETWORK DIGITAL VIDEO MANAGEMENT SYSTEM
PRINT
PLAY/EDIT
ANALOG/STORED
MONITOR
ANALOG/LIVE

FIG. 1

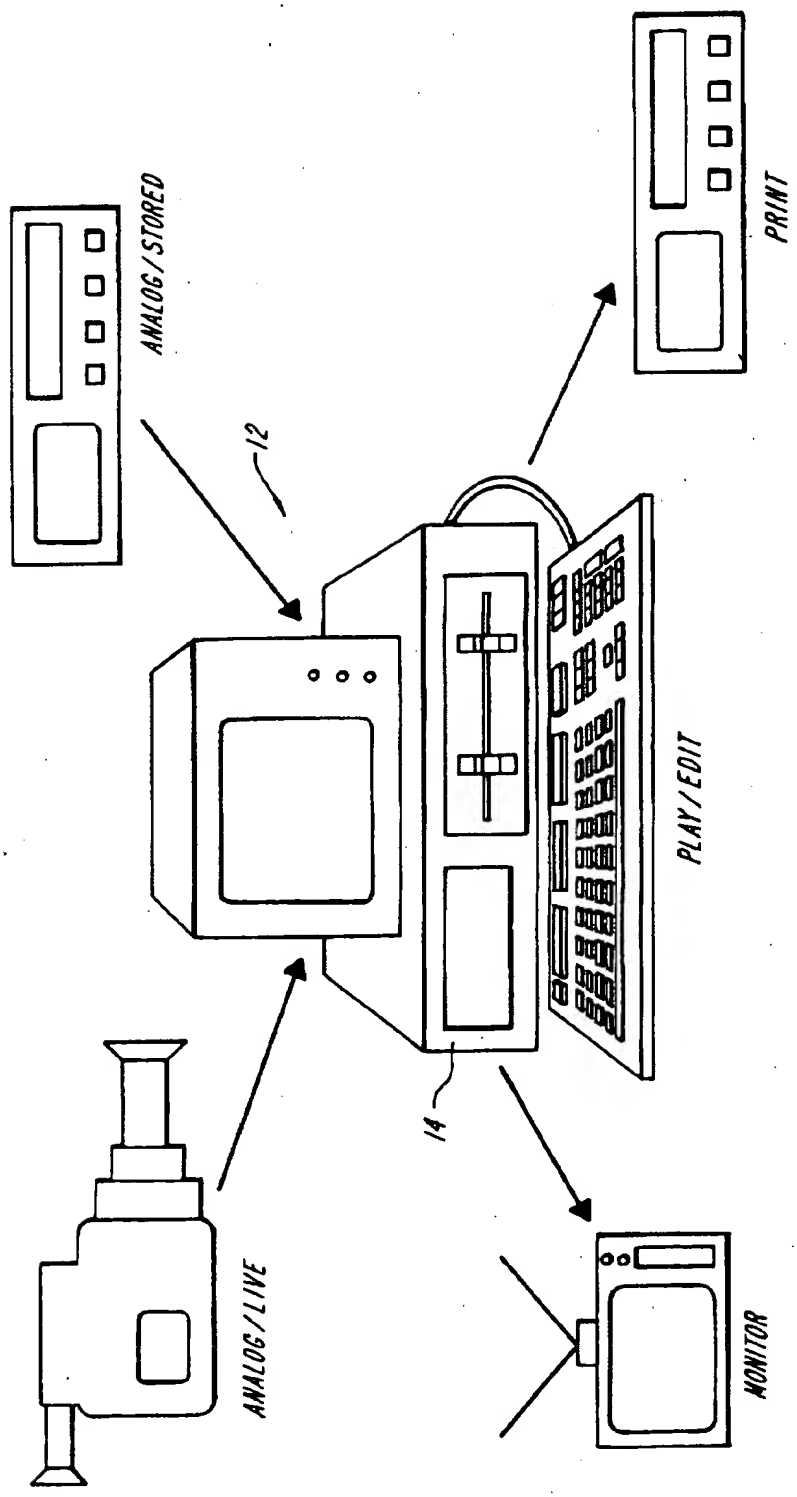


FIG. 2

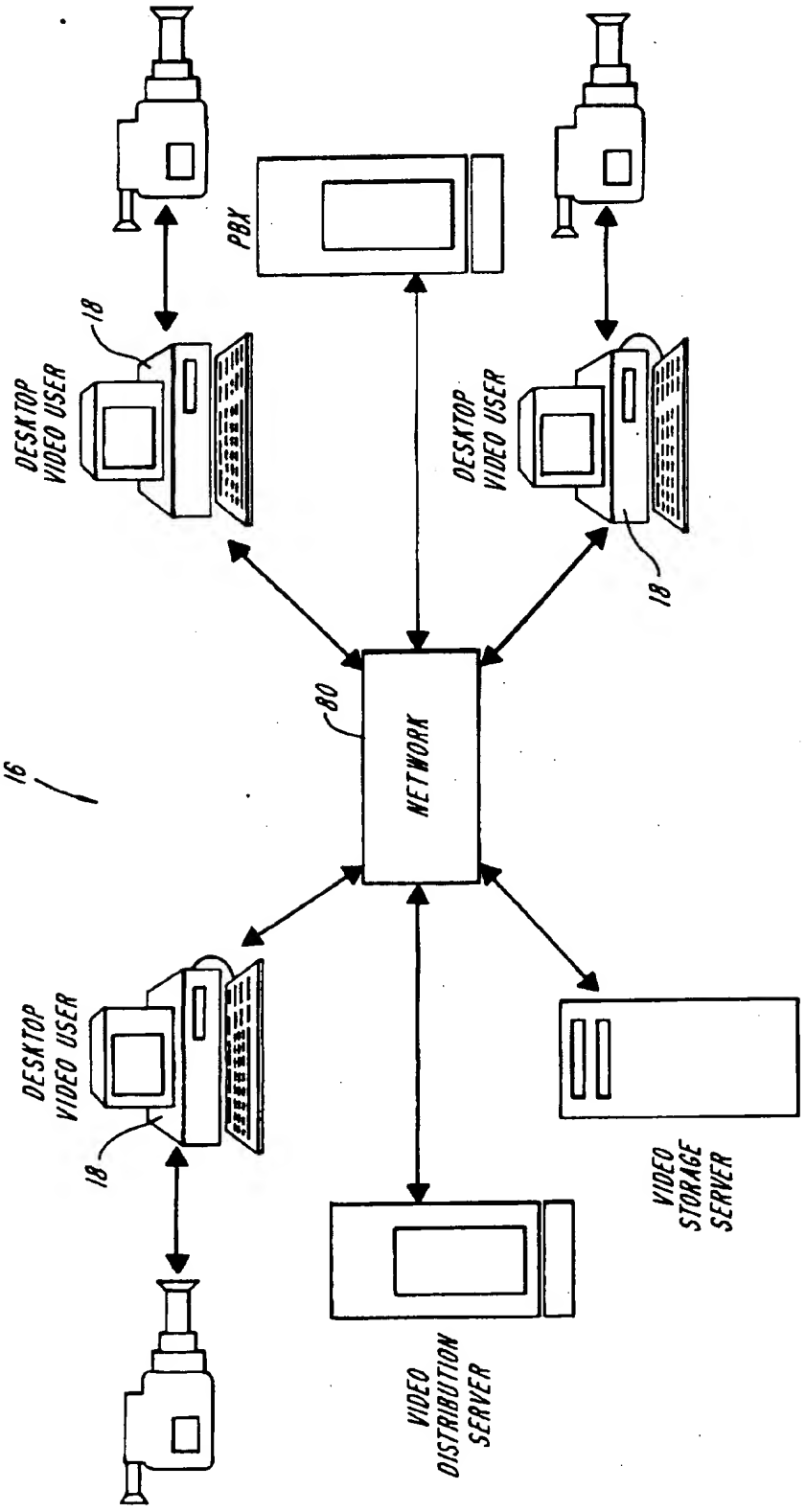

PBX
18
DESKTOP VIDEO USER
DESKTOP VIDEO USER
18
80
NETWORK
16
DESKTOP VIDEO USER
18
VIDEO STORAGE SERVER
VIDEO DISTRIBUTION SERVER

FIG. 3

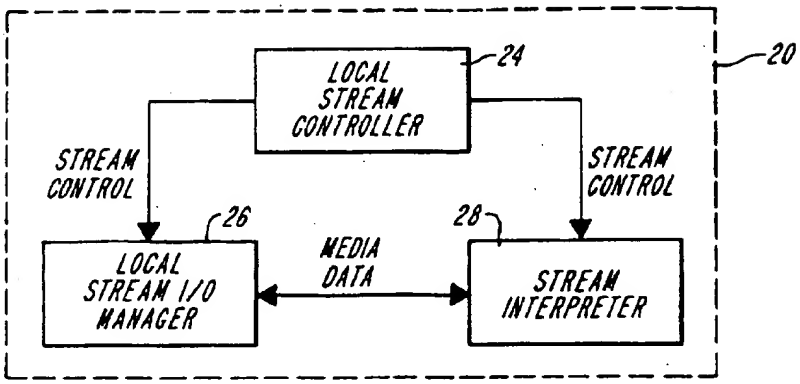

LOCAL STREAM CONTROLLER
24
20
STREAM CONTROL
STREAM CONTROL
26
28
LOCAL STREAM I/O MANAGER
MEDIA DATA
STREAM INTERPRETER

FIG. 4

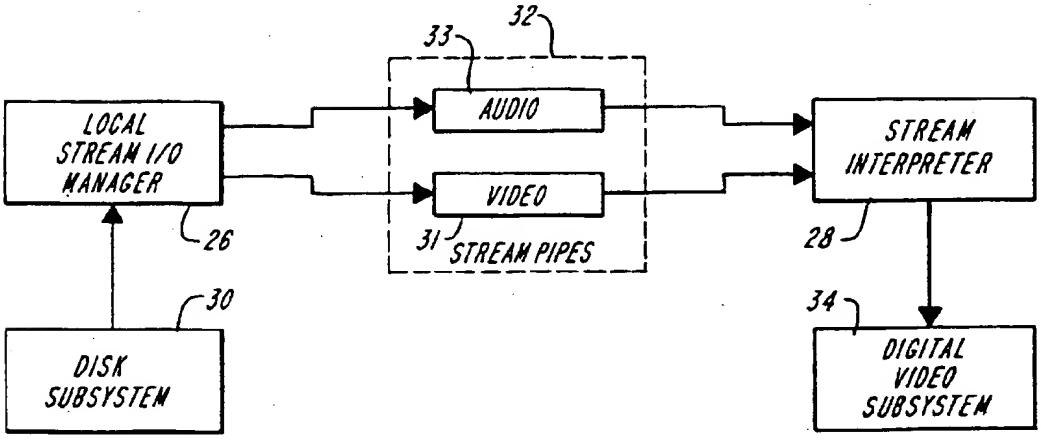

32
33
AUDIO
VIDEO
31
STREAM PIPES
LOCAL STREAM I/O MANAGER
26
30
DISK SUBSYSTEM
STREAM INTERPRETER
28
34
DIGITAL VIDEO SUBSYSTEM

FIG. 5

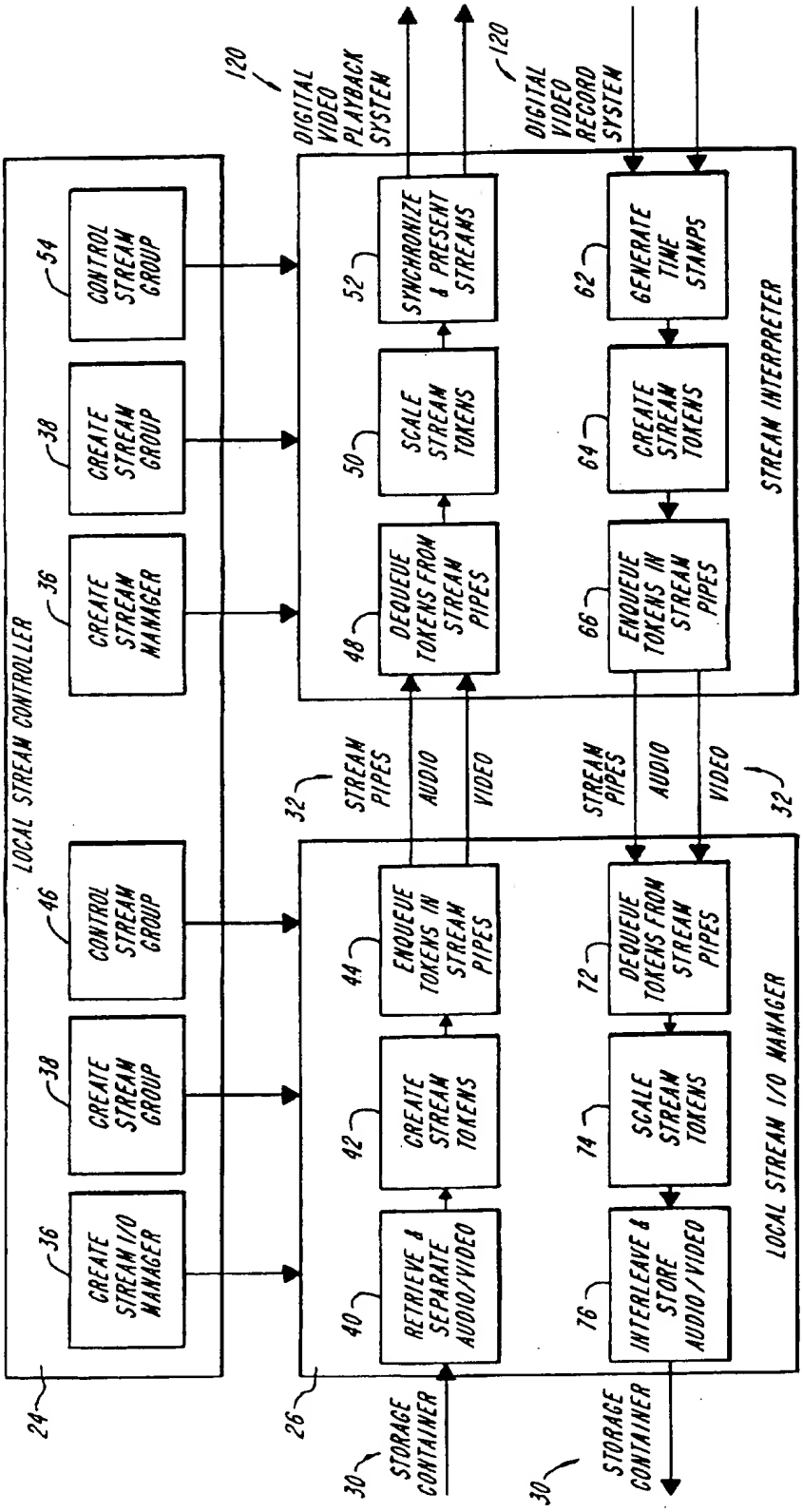

LOCAL STREAM CONTROLLER
CREATE STREAM I/O MANAGER
CREATE STREAM GROUP
CONTROL STREAM GROUP
CREATE STREAM MANAGER
CREATE STREAM GROUP
CONTROL STREAM GROUP
DIGITAL VIDEO PLAYBACK SYSTEM
DIGITAL VIDEO RECORD SYSTEM
SYNCHRONIZE & PRESENT STREAMS
SCALE STREAM TOKENS
DEQUEUE TOKENS FROM STREAM PIPES
GENERATE TIME STAMPS
CREATE STREAM TOKENS
ENQUEUE TOKENS IN STREAM PIPES
STREAM INTERPRETER
STREAM PIPES
AUDIO
VIDEO
STREAM PIPES
AUDIO
VIDEO
ENQUEUE TOKENS IN STREAM PIPES
CREATE STREAM TOKENS
RETRIEVE & SEPARATE AUDIO/VIDEO
DEQUEUE TOKENS FROM STREAM PIPES
SCALE STREAM TOKENS
INTERLEAVE & STORE AUDIO/VIDEO
LOCAL STREAM I/O MANAGER
STORAGE CONTAINER
STORAGE CONTAINER
24
26
30
30
32
32
36
38
46
54
36
38
48
50
52
62
64
66
40
42
44
72
74
76
120
120

FIG. 6

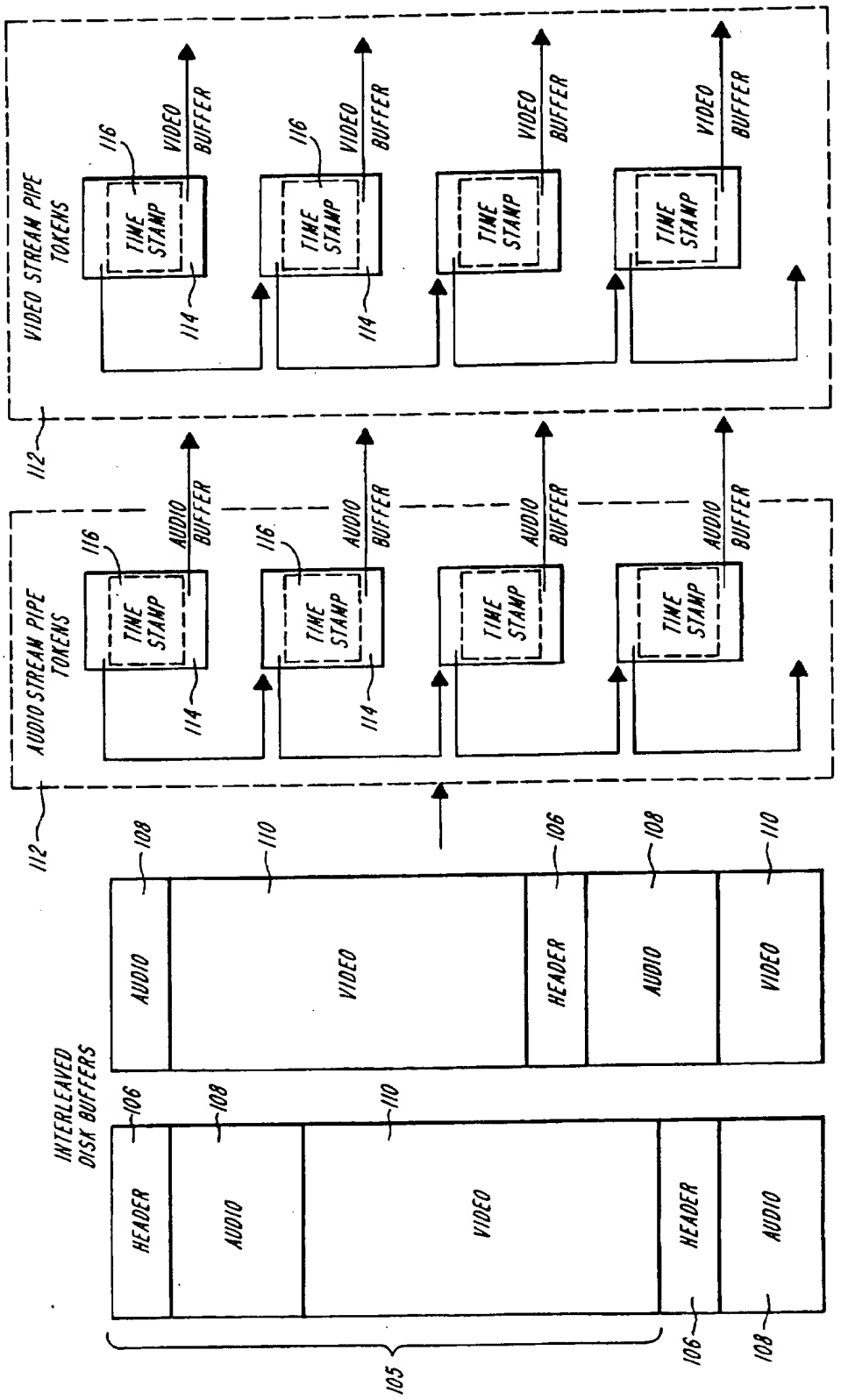

VIDEO STREAM PIPE TOKENS
AUDIO STREAM PIPE TOKENS
TIME STAMP
VIDEO BUFFER
AUDIO BUFFER
116
114
112
INTERLEAVED DISK BUFFERS
AUDIO
VIDEO
HEADER
106
108
110
105

FIG. 7

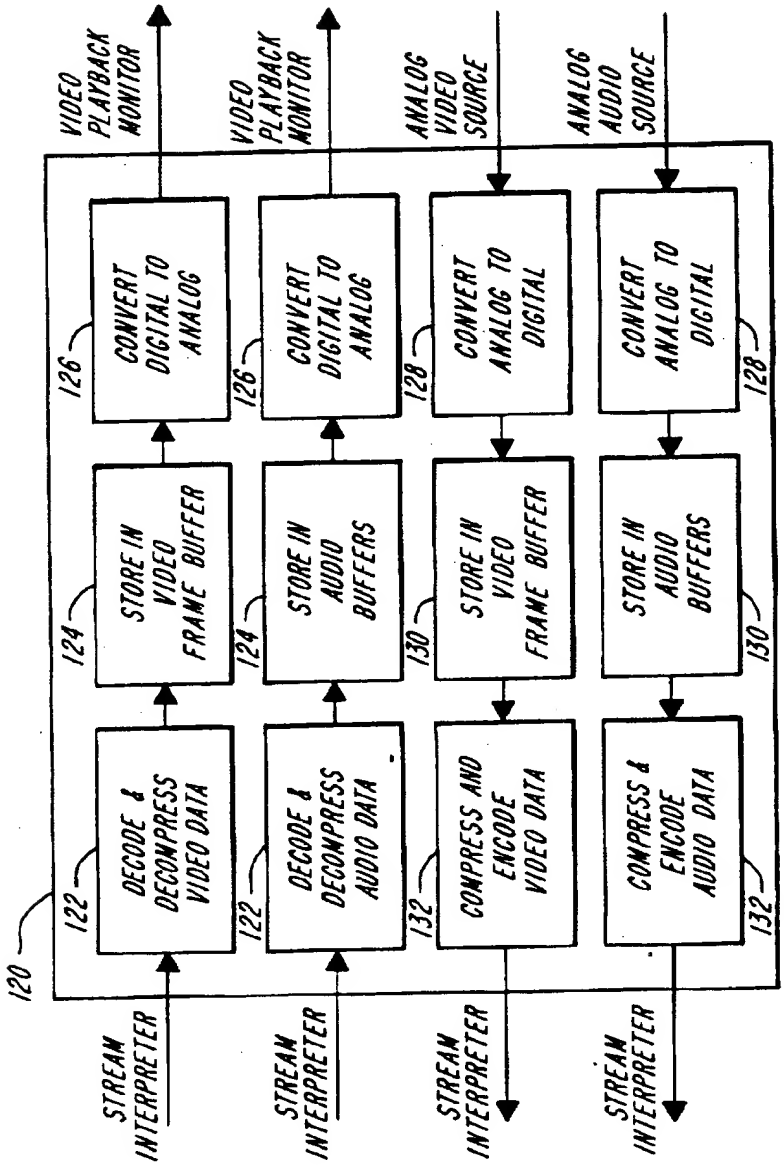

VIDEO PLAYBACK MONITOR
VIDEO PLAYBACK MONITOR
ANALOG VIDEO SOURCE
ANALOG AUDIO SOURCE
CONVERT DIGITAL TO ANALOG
CONVERT DIGITAL TO ANALOG
CONVERT ANALOG TO DIGITAL
CONVERT ANALOG TO DIGITAL
STORE IN VIDEO FRAME BUFFER
STORE IN AUDIO BUFFERS
STORE IN VIDEO FRAME BUFFER
STORE IN AUDIO BUFFERS
DECODE & DECOMPRESS VIDEO DATA
DECODE & DECOMPRESS AUDIO DATA
COMPRESS AND ENCODE VIDEO DATA
COMPRESS & ENCODE AUDIO DATA
STREAM INTERPRETER
STREAM INTERPRETER
STREAM INTERPRETER
STREAM INTERPRETER
120
122
122
132
132
124
124
130
130
126
126
128
128

FIG. 8

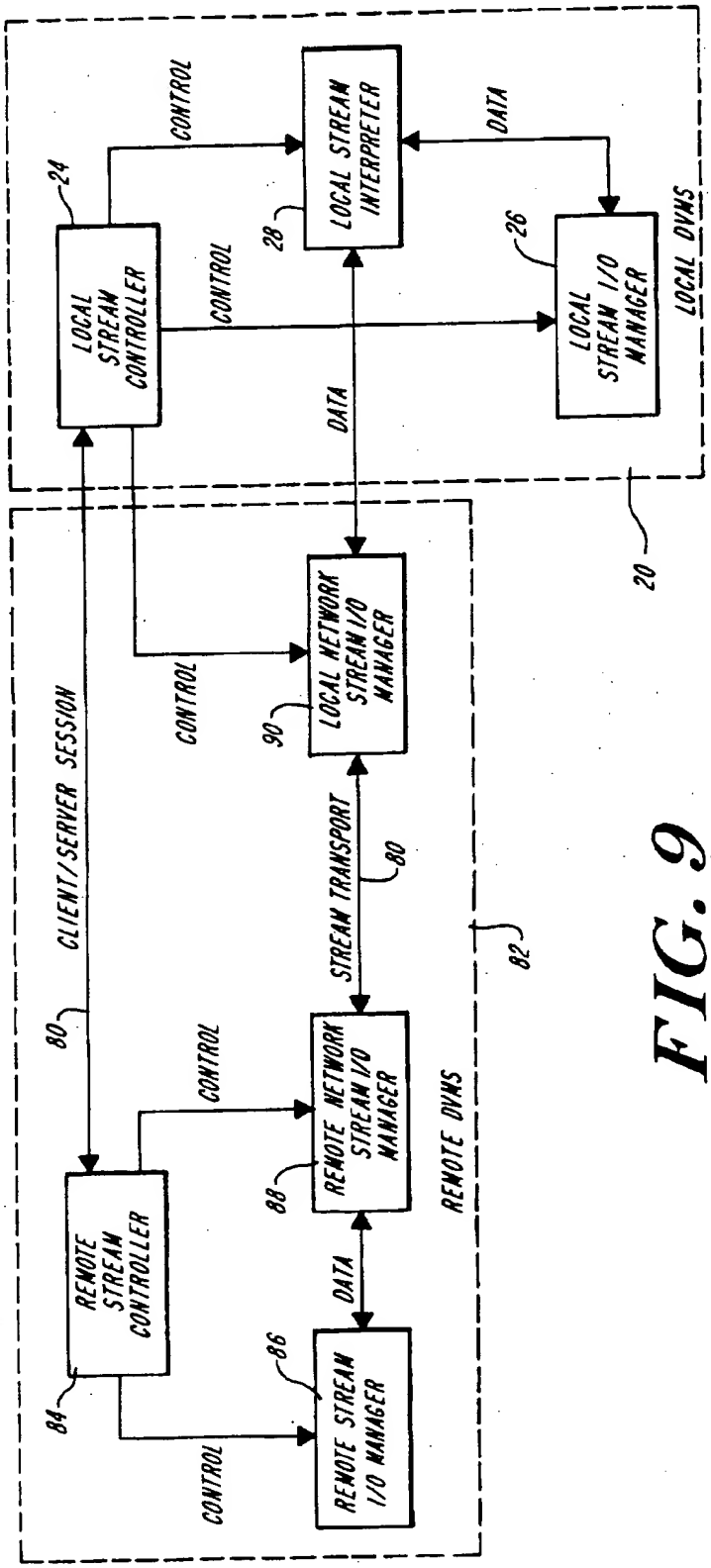

LOCAL DVWS
LOCAL STREAM INTERPRETER
LOCAL STREAM CONTROLLER
LOCAL STREAM I/O MANAGER
LOCAL NETWORK STREAM I/O MANAGER
REMOTE DVWS
REMOTE NETWORK STREAM I/O MANAGER
REMOTE STREAM CONTROLLER
REMOTE STREAM I/O MANAGER
CONTROL
DATA
CLIENT/SERVER SESSION
STREAM TRANSPORT
20
24
26
28
80
82
84
86
88
90

FIG. 9

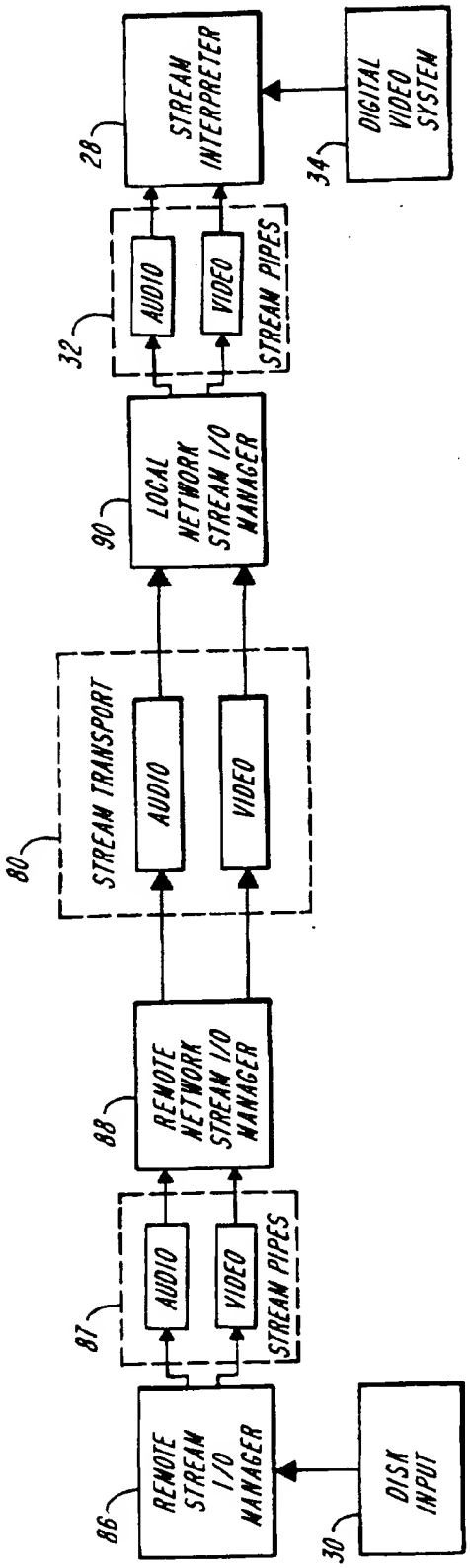

DISK INPUT
30
REMOTE STREAM I/O MANAGER
86
STREAM PIPES
87
AUDIO
VIDEO
REMOTE NETWORK STREAM I/O MANAGER
88
STREAM TRANSPORT
80
AUDIO
VIDEO
LOCAL NETWORK STREAM I/O MANAGER
90
STREAM PIPES
32
AUDIO
VIDEO
STREAM INTERPRETER
28
DIGITAL VIDEO SYSTEM
34

FIG. 10

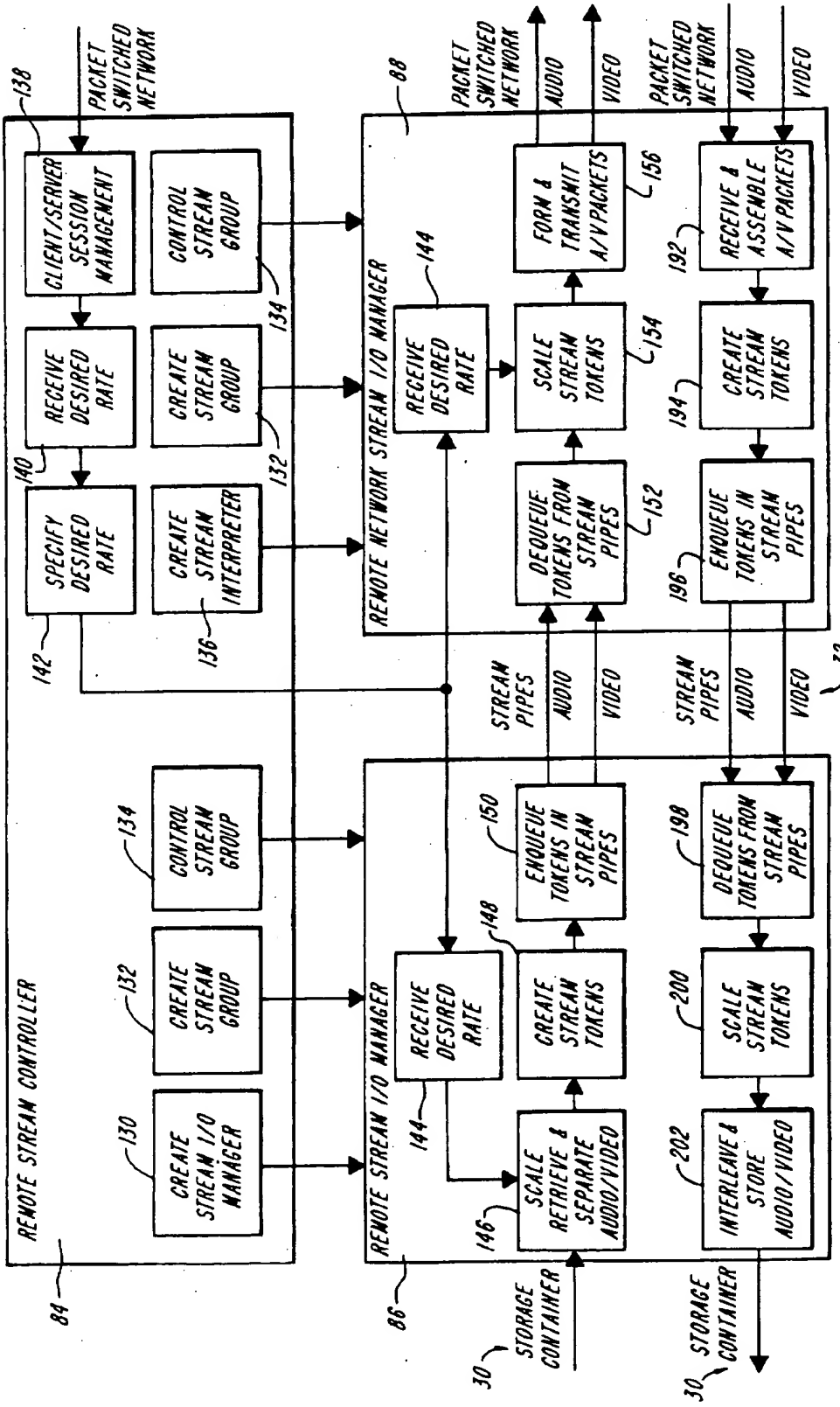

REMOTE STREAM CONTROLLER
84
CREATE STREAM I/O MANAGER
130
CREATE STREAM GROUP
132
CONTROL STREAM GROUP
134
SPECIFY DESIRED RATE
140
142
CREATE STREAM INTERPRETER
136
REMOTE STREAM I/O MANAGER
86
RECEIVE DESIRED RATE
144
CREATE STREAM TOKENS
148
ENQUEUE TOKENS IN STREAM PIPES
150
SCALE RETRIEVE & SEPARATE AUDIO/VIDEO
146
DEQUEUE TOKENS FROM STREAM PIPES
152
SCALE STREAM TOKENS
154
FORM & TRANSMIT A/V PACKETS
156
STREAM PIPES
AUDIO
VIDEO
32
STORAGE CONTAINER
30
CLIENT/SERVER SESSION MANAGEMENT
138
PACKET SWITCHED NETWORK
REMOTE NETWORK STREAM I/O MANAGER
88
RECEIVE DESIRED RATE
144
CREATE STREAM GROUP
132
CONTROL STREAM GROUP
134
CREATE STREAM INTERPRETER
136
DEQUEUE TOKENS FROM STREAM PIPES
152
SCALE STREAM TOKENS
154
FORM & TRANSMIT A/V PACKETS
156
PACKET SWITCHED NETWORK
AUDIO
VIDEO
RECEIVE DESIRED RATE
140
SPECIFY DESIRED RATE
142
CREATE STREAM TOKENS
194
ENQUEUE TOKENS IN STREAM PIPES
196
RECEIVE & ASSEMBLE A/V PACKETS
192
DEQUEUE TOKENS FROM STREAM PIPES
198
SCALE STREAM TOKENS
200
INTERLEAVE & STORE AUDIO/VIDEO
202
STREAM PIPES
AUDIO
VIDEO
32
STORAGE CONTAINER
30

FIG. 11A

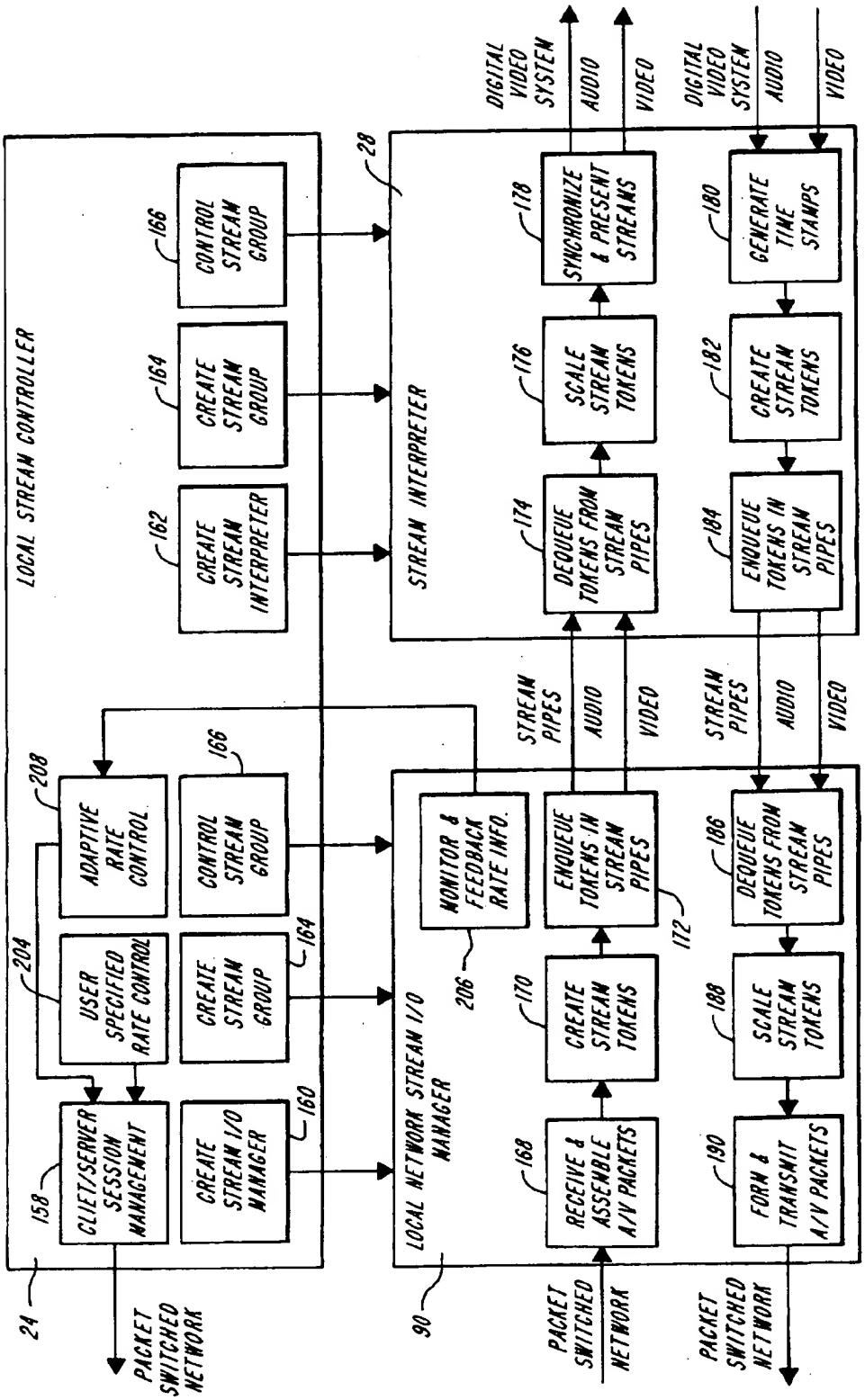

LOCAL STREAM CONTROLLER
CLIET/SERVER SESSION MANAGEMENT
158
CREATE STREAM I/O MANAGER
160
USER SPECIFIED RATE CONTROL
204
ADAPTIVE RATE CONTROL
208
CREATE STREAM GROUP
164
CONTROL STREAM GROUP
166
CREATE STREAM INTERPRETER
162
CREATE STREAM GROUP
164
CONTROL STREAM GROUP
166
24
PACKET SWITCHED NETWORK
28
STREAM INTERPRETER
LOCAL NETWORK STREAM I/O MANAGER
90
RECEIVE & ASSEMBLE A/V PACKETS
168
CREATE STREAM TOKENS
170
ENQUEUE TOKENS IN STREAM PIPES
172
MONITOR & FEEDBACK RATE INFO.
206
DEQUEUE TOKENS FROM STREAM PIPES
174
SCALE STREAM TOKENS
176
SYNCHRONIZE & PRESENT STREAMS
178
PACKET SWITCHED NETWORK
FORM & TRANSMIT A/V PACKETS
190
SCALE STREAM TOKENS
188
DEQUEUE TOKENS FROM STREAM PIPES
186
ENQUEUE TOKENS IN STREAM PIPES
184
CREATE STREAM TOKENS
182
GENERATE TIME STAMPS
180
STREAM PIPES
AUDIO
VIDEO
STREAM PIPES
AUDIO
VIDEO
DIGITAL VIDEO SYSTEM
AUDIO
VIDEO
DIGITAL VIDEO SYSTEM
AUDIO
VIDEO

FIG. 11B

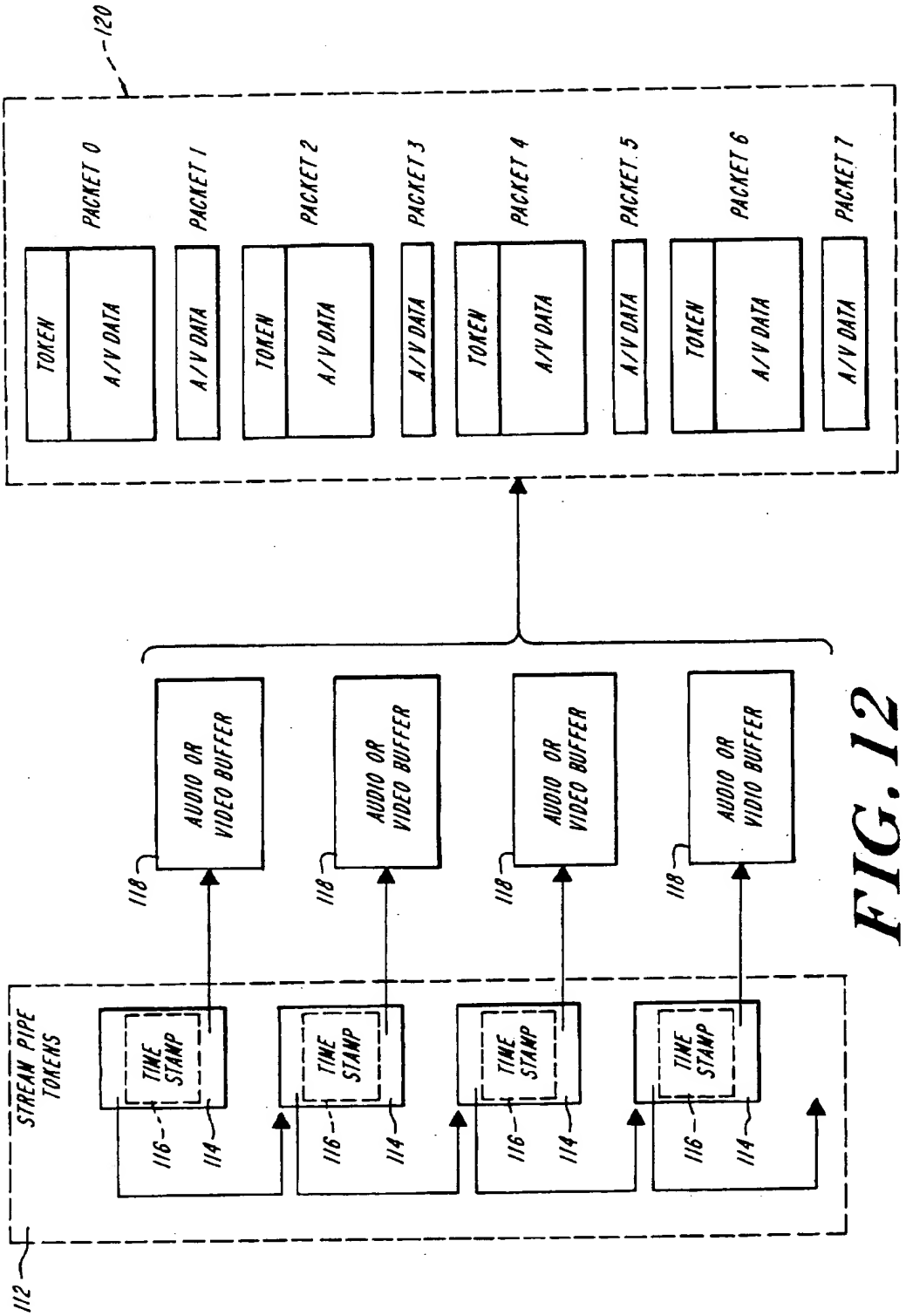

120
PACKET 0
PACKET 1
PACKET 2
PACKET 3
PACKET 4
PACKET 5
PACKET 6
PACKET 7
TOKEN
A/V DATA
AUDIO OR VIDEO BUFFER
118
STREAM PIPE TOKENS
TIME STAMP
116
114
112

FIG. 12

DIGITAL MEDIA DATA STREAM NETWORK MANAGEMENT SYSTEM

BACKGROUND OF THE INVENTION

This invention relates to the management of digitized media stream data, e.g., digitized video, and particularly relates to the capture, storage, distribution, access and presentation of digital video within a network computing environment.

Extensive technological advances in microelectronics and digital computing systems have enabled digitization of a wide range of types of information; for example, digital representations of text, graphics, still images and audio are now in widespread use. Advances in compression, storage, transmission, processing and display technologies have recently provided the capabilities required to extend the field of digitization to additionally include video information.

Conventionally, digitized audio and video are presented on, for example, a computer system or network by capturing and storing the audio and video streams in an interleaved fashion, i.e., segments of the two streams are interleaved. This requires storage of the digital audio and video in a single stream storage container, and further requires retrieving chunks of interleaved audio and video data at an aggregate rate which matches the nominal rate of an active presentation sequence. In this way, one unit of video (say, a frame) is physically associated in storage with one unit of audio (say, a corresponding 33 msec clip), and the two are retrieved from storage as a unit. Sequences of such audio and video units are then provided to a presentation and decoder digital subsystem in an alternating fashion, whereby each audio and video unit of a pair is provided in sequence.

Computer systems that provide this audio and video management functionality typically include digital compression/decompression and capture/presentation hardware and software, and digital management system software, all of which is based upon and depends upon the interleaved format of the audio and video streams it processes.

Currently, handling of audio and video in a network environment is also based on a scheme in which capture, storage, and transmission of audio and video must be carried out using interleaved audio and video streams. This interleaving extends to the transmission of audio and video streams across the network in an interleaved format within transmission packets.

Synchronization of audio with video during an active presentation sequence is conventionally achieved by initially interleaving the audio and video streams in storage and then presenting audio and video chunks at the nominal rate specified for an active presentation sequence.

In "Time Capsules: An Abstraction for Access to continuous-Media Data," by Herrtwich, there is disclosed a frame-work based on time capsules to describe how timed data shall be stored, exchanged, and accessed in real-time systems. When data is stored into such a time capsule, a time stamp and a duration value are associated with the data item. The time capsule abstraction includes the notion of a dock for ensuring periodic data access that is typical for continuous-media applications. By modifying the parameters of a dock, presentation effects such as time lapses or slow motion may be achieved.

While the Herrtwich disclosure provides a time capsule abstraction for managing time-based data, the disclosure does not provide any technique for synchronizing time-based data based on the time capsule abstraction, and does not address the requirements of time-based data management in a network environment. Furthermore, the disclosure does not address processing of time-based data streams as a function of their interleaved format or manipulation of that format.

SUMMARY OF THE INVENTION

In general, in one aspect, the invention features a computer-based media data processor for controlling the computer presentation of digitized continuous time-based media data composed of a sequence of presentation units. Each presentation unit is characterized by a prespecified presentation duration and presentation time during a computer presentation of the media data and is further characterized as a distinct media data type. In the processor of the invention, a media data input manager retrieves media data from a computer storage location in response to a request for computer presentation of specified presentation unit sequences, and determines the media data type of each presentation unit in the retrieved media data. The input manager then designates each retrieved presentation unit to a specified media data presentation unit sequence based on the media data type determination for that presentation unit. The input manager then assembles a sequence of presentation descriptors for each of the specified presentation unit sequences, each descriptor comprising media data for one designated presentation unit in that sequence, and each sequence of presentation descriptors being of a common media data type; and then associates each presentation descriptor with a corresponding presentation duration and presentation time, based on the retrieved media data. Finally, the input manager links the presentation descriptors of each sequence to establish a progression of presentation units in that sequence.

A media data interpreter of the invention indicates a start time of presentation processing of the presentation descriptor sequences, and accordingly, maintains a current presentation time as the sequences are processed for presentation. The interpreter counts each presentation unit in the media data sequences after that unit is processed for presentation, to maintain a distinct current presentation unit count for each sequence, and compares for each of the presentation unit sequences a product of the presentation unit duration and the current presentation unit count of that sequence with the current presentation time after each presentation unit from that sequence is processed for presentation. Based on the comparison, the interpreter releases a presentation unit next in that presentation unit sequence to be processed for presentation when the product matches the current presentation time count, and deletes a presentation unit next in that presentation unit sequence when the product exceeds the current presentation time count.

In general, in another aspect, the invention features a media data processor for controlling transmission of digitized media data in a packet switching network. Such a network comprises a plurality of client computer processing nodes interconnected via packet-based data distribution channels. In the invention, a remote media data controller receives from a client processing node a request for presentation of specified presentation unit sequences, and in response to the request, retrieves media data from a corresponding media access location. A remote media data input manager of the invention then determines the media data type of each presentation unit in the retrieved media data, and designates each retrieved presentation unit to a specified

media data presentation unit sequence based on the media data type determination for that presentation unit. Then the input manager assembles a sequence of presentation descriptors for each of the specified presentation unit sequences, each descriptor comprising media data for one designated presentation unit in that sequence, and all presentation descriptors in an assembled sequence being of a common media data type. The interpreter associates each presentation descriptor with a corresponding presentation duration and presentation time, based on the retrieved media data; and finally, links the descriptors in each assembled sequence to establish a progression of presentation units in each of the specified presentation unit sequences.

A remote network media data manager of the invention assembles transmission presentation unit packets each composed of at least a portion of a presentation descriptor and its media data, all presentation descriptors and media data in an assembled packet being of a common media data type; and releases the assembled packets for transmission via the network to the client processing node requesting presentation of the specified presentation unit sequences.

A local media data controller of the invention transmits the presentation unit sequence request to the remote media data controller from the client processing node, and controls starting and stopping of sequence presentation in response to user specifications.

A local network media data manager of the invention receives at the client processing node the transmission presentation unit packets via the network, and designates a presentation unit sequence for each presentation descriptor and its media data in the received packets to thereby assemble the presentation descriptor sequences each corresponding to one specified presentation unit sequence, all presentation descriptors in an assembled sequence being of a common media data type. Then the local network media data manager links the descriptors in each assembled sequence to establish a progression of presentation units for each of the presentation unit sequences.

A local media data interpreter of the invention accepts the assembled presentation descriptor sequences one descriptor at a time and releases the sequences for presentation one presentation unit at a time. In this process, the local interpreter indicates a start time of presentation processing of the sequences, and accordingly, maintains a current presentation time as the descriptor sequences are processed for presentation. Based on the presentation duration of each presentation unit, the interpreter synchronizes presentation of the specified presentation unit sequences with the current presentation time.

In preferred embodiments, the specified media data presentation unit sequences comprise a video frame sequence including a plurality of intracoded video frames; preferably, each frame of the video frame sequence comprises an intracoded video frame, and more preferably, the video frame sequence comprises a motion JPEG video sequence and an audio sequence. In other preferred embodiments, each of the plurality of intracoded video frames comprises a key frame and is followed by a plurality of corresponding non-key frames, each key frame including media data information required for presentation of the following corresponding non-key frames.

In other preferred embodiments, synchronization of presentation of the specific presentation unit sequences is accomplished by the local media data interpreter by comparing for each of the presentation descriptors in each of the presentation descriptor sequences the presentation time corresponding to that descriptor with the currently maintained presentation time. Based on this comparison, the interpreter releases a next sequential presentation unit to be processed for presentation when the corresponding presentation time of that descriptor matches the current presentation time, and deletes a next sequential presentation unit to be processed for presentation when the current presentation time exceeds the corresponding presentation time of that descriptor.

In other preferred embodiments, synchronization of presentation of the specific presentation unit sequences is accomplished by the local media data interpreter by counting each presentation descriptor in the sequences after that presentation unit is released to be processed for presentation, to maintain a distinct current presentation unit count for each sequence. Then, the interpreter compares for each of the presentation unit sequences a product of the presentation unit duration and the current presentation descriptor count of that sequence with the currently maintained presentation time after a presentation unit from that sequence is released to be processed for presentation. Based on the comparison, the interpreter releases a next sequential presentation unit in that presentation unit sequence when the product matches the currently maintained presentation time, and deletes a next sequential presentation unit in that presentation unit sequence when the product exceeds the currently maintained presentation time.

In other preferred embodiments, the remote media data controller of the invention receives from the local media data controller, via the network, an indication of a specified presentation data rate at which the specified presentation unit sequences are to be transmitted via the network to the client node. The media data retrieved comprises a plurality of storage presentation unit sequences stored in a computer storage location, each storage presentation unit sequence composed of presentation units corresponding to a specified presentation unit sequence and all presentation units in a storage presentation unit sequence being of a common media data type. The remote media data input manager designates each of a portion of the presentation unit descriptors as the descriptor sequences are assembled, the portion including a number of descriptors based on the specified presentation data rate, each designated descriptor comprising null media data, to thereby compose the presentation descriptor sequences with only a portion of storage presentation unit media data. With this designation, the specified presentation unit sequences attain the specified presentation data rate of transmission.

In the invention, the separation of media streams and distinctly formatting of network transmission packets for each stream provides an opportunity and the facility to examine, process, and make transmission decisions about each stream and each presentation unit independent of other streams and presentation units. As a result, the media processor of the invention can make presentation decisions about a given presentation unit independent of the other units in the corresponding stream, and can make those decisions "on-the-fly". This capability provides for real time scaling and network load adjustment as a stream is retrieved, processed, and transmitted across the network.

Further aspects, features, and advantages of the invention are set forth in the following specification and the claims.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 is a schematic diagram of media stream access and delivery points with which the digital video management system of the invention may interface;

FIG. 2 is a schematic diagram of a stand-alone implementation of the digital video management system of the invention;

FIG. 3 is a schematic diagram of a network implementation of the digital video management system of the invention;

FIG. 4 is a schematic diagram of the local digital video management system manager modules of the invention;

FIG. 5 is a schematic diagram illustrating the flow of media stream data between the stream I/O manager and stream interpreter modules of the local digital video management system manager of FIG. 4;

FIG. 6 is a schematic flow chart illustrating presentation and capture scenarios carried out by the local digital video management system manager of FIG. 4;

FIG. 7 is a schematic illustration of the translation from media stream storage format to token format carried out by the local digital video management system manager of FIG. 4;

FIG. 8 is a schematic flow chart illustrating presentation and capture scenarios carried out by a digital video system used in conjunction with the local digital video management system manager scenarios of FIG. 6;

FIG. 9 is a schematic diagram of the local digital video management system manager and the remote digital video management manager modules of the invention in a network implementation;

FIG. 10 is a schematic diagram illustrating the flow of media stream data between the remote and local digital video management manager modules of the invention in a network implementation;

FIG. 11A is a schematic flow chart illustrating presentation and capture scenarios carried out by the remote digital video management system manager of FIG. 9;

FIG. 11B is a schematic flow chart illustrating presentation and capture scenarios carried out by the local digital video management system manager of FIG. 9;

FIG. 12 is a schematic illustration of the translation of stream tokens of FIG. 7 into packet format.

DESCRIPTION OF A PREFERRED EMBODIMENT

Referring to FIG. 1, there is illustrated the digital video management system (DVMS) 10 of the invention. The DVMS provides the ability to capture, store, transmit, access, process and present live or stored media stream data, independent of its capture or storage location, in either a stand-alone or a network environment. The DVMS accommodates media stream data, i.e., continuous, high data-rate, real-time data, including video, audio, animation, photographic stills, and other types of continuous, time-based media data. Throughout this description, the DVMS of the invention will be explained with reference to audio and video streams, but it must be remembered that any time-based media data stream may be managed in the system. In the DVMS, as shown in FIG. 1, media data may be accessed from, e.g., live analog capture, analog or digital file storage, or live digital capture from, e.g., a PBX (private branch exchange) server, among other access points. The accessed media is managed by the DVMS for delivery to, e.g., a presentation monitor, a computer system for editing and presentation on the computer, a VCR tape printer, or digital storage, or sent to a PBX server.

Of great advantage, the DVMS management scheme is independent of any particular storage or compression technology used to digitize the data streams, and further, is independent of any particular communication protocols or delivery platform of a network in which the DVMS is implemented. Additionally, the DVMS is industry standards-based yet is flexible and standards-extensible, via its layered architecture, which incorporates multiple management platforms. Each of these features and advantages will be explained in detail in the discussion to follow.

Digital Video Management System Components

The DVMS of the invention is based on a technique whereby media data streams are handled and managed as distinct and separate media data streams in which there is no interleaving of media data. Here the term "stream" is meant to represent a dynamic data type, like video, as explained above, and thus, a stream consists of dynamic information that is to be produced and consumed in a computer system or network with temporal predictability. A stream contains a succession of sequences. Sequences can themselves contain sequences; in turn, each sequence contains a succession of segments. Streams, sequences and segments, as information identifiers, have no media type-specific semantics. Rather, they are convenient abstractions for specifying and organizing dynamic data types to be managed by the management system of the invention. An easily understood analogy to streams, sequences and segments is that of documents containing chapters, sections and sentences.

Streams are characterized by their media data type, e.g., audio, video, or animation data types. Sequences represent information that is meaningful to the user. For example, a video sequence may represent a video clip containing a video scene. Segments can be convenient "chunks" of data for editing and mixing that data. Segments may also represent units of data that are temporally linked, as when using a video compression scheme that produces key video frames and corresponding following difference video frames.

In the DVMS of the invention, streams that are intended for synchronous presentation can be grouped into a stream group of distinct constituent streams (i.e., without interleaving). Although constituent streams in such a stream group may be stored in an interleaved form within a storage container, the DVMS can dynamically coordinate separately stored streams; in either case, the system processes the streams distinctly, rather than in an interleaved fashion.

Segments of streams contain presentation units. A presentation unit is a unit of continuous, temporally-based data to be presented, and accordingly, has an associated presentation time and presentation duration. A presentation time indicates the appropriate point in the sequence of a presentation at which the associated presentation unit is to be played, relative to a time base for the ongoing presentation. A presentation duration indicates the appropriate interval of time over which the associated presentation unit is to be played in the ongoing presentation. Thus, a video presentation unit comprises a video frame, and an audio presentation unit comprise a number of sound samples associated with a frame duration.

As mentioned above, the DVMS may be implemented in a stand-alone computer system or a computer-based, packet switched network. Referring to FIG. 2, in a stand-alone computer system implementation 12, live or stored media streams are accessed and captured for presentation and editing on the stand-alone computer 14. The captured, and optionally edited media streams may then be delivered to a presentation monitor or to a VCR tape printer utility.

Referring to FIG. 3, a packet switching network in which the DVMS is implemented comprises desktop computer systems 18 which are linked via a packet switching network

80, which is controlled by the DVMS network implementation 16. The network 80 may comprise a local area network (LAN) or a wide area network (WAN), or a combination of one or more LANs and WANs. The DVMS provides access to and capture of media streams from live analog video capture, e.g., a VCR or camcorder, a network, storage or PBX server, or one of the desktop computers, and in turn manages the transmission of the media stream data across the network back to any of the access points.

The digital video management system consists of a local DVMS manager and a remote DVMS manager. The local DVMS manager provides a client operating environment, and thus resides on a stand-alone computer or each client computer in a network, "client" here being defined as a computer system or one of the access points in a network that request media data; the remote DVMS manager provides a network operating environment, and thus resides on a network server. The local DVMS manager may be implemented on, for example, IBM-compatible personal computers running Microsoft® Windows™, to thereby provide high-level, industry-standard access to underlying digital video services. This local DVMS manager implementation may support, for example, the industry-standard Microsoft®) digital video MCI API for application development. The local DVMS manager incorporates an efficient data-flow subsystem, described below, that is highly portable to other operating systems.

The DVMS system of the invention is preferably implemented as an application programming interface suite that includes interfaces for a computer programming application to include media data stream management capability within the application. Thus, the DVMS interfaces with an underlying programming application via interface calls that initiate media data stream functions within the realm of the programming application. Such an interface implementation will be understandable to those skilled in the art of C programming.

The remote DVMS manager acts to dynamically link a client and a server in the packet network environment. The architecture of this manager has the important advantage of supporting the ability to scale distinct, noninterleaved media data streams, as discussed in depth below. This ability to scale packet-based video, thereby creating scalable packet video, is a facility which permits adaptive bandwidth management for dynamic media data types in both LANs and WANs. The remote DVMS manager may be implemented as a Netware© Loadable Module, on, for example, the Novell Netware© operating system.

Local DVMS Manager

The local DVMS manager manages the access and capture of media data streams transparently, i.e., without impacting the functionality of the application program which requested that access and capture. The local DVMS manager works with a digital video system, implemented either in special purpose digital video hardware or in special purpose software-based emulation of the digital hardware.

Referring to FIG. 4, the local DVMS manager 20 consists of three modules: the stream controller 24, stream input/output (I/O) manager 26, and the stream interpreter 28. This modularity is exploited in the DVMS design to separate the flow of data in a media data streams from the flow of control information for that media stream through the system. Based on this data and control separation, streams data and stream control information are each treated as producing distinct interactions among the three manager modules, which operate as independent agents. The I/O manager, interpreter and controller agents are each mapped via the local DVMS manager to independently scheduable operating system processes with independent program control flow and data space allocation. The flow of media stream data is managed by the stream I/O manager 26 and the stream interpreter 28, while the flow of control information is managed by the stream controller 24. Each of these management functions is explained in detail below.

The stream I/O manager module 26 is responsible for the dynamic supply of media data streams, e.g., audio and video streams, from or to the stream interpreter. This module also provides efficient file format handling functions for the media data., if it is accessed via a storage file, e.g., a DVI® AVSS file. In a stand-alone implementation of the DVMS of the invention, the stream I/O manager provides retrieval and storage of media data streams from or to points of media access, such as digital or analog storage containers, while in a network implementation of the DVMS, as described below, the remote DVMS manager modules provide retrieval and storage at points of media access via the network. Most importantly, the stream I/O manager performs a translation from the representation of audio and video information as that information is stored to the corresponding dynamic computer-based representation. This translation is explained in detail below.

The stream interpreter module 28 is responsible for managing the dynamic computer-based representation of audio and video as that representation is manipulated in a stand-alone computer or a computer linked into a packet network. This dynamic management includes synchronization of retrieved audio and video streams, and control of the rate at which the audio and video information is presented during a presentation sequence. In addition, the stream interpreter module manages the capture, compression, decompression and playback of audio and video information. This module is, however, compression technology-independent and additionally is device-independent. Base services of a digital video subsystem, including, for example, hardware for capture and presentation functions, are preferably implemented to be accessed through a standard API suite of digital video primitives, which encapsulate any functions unique to a particular compression or device technology.

The following suite of primitive functions provide device-independent access to the base services of a digital video subsystem:

Open: Open a specified device, initialize it, and return a handle for further requests;

Close: Close a specified device and free up any associated resources; Get_Capabilities: Query a device's capabilities, e.g., display resolutions, compression format, etc.;

Start: Start decoding and displaying data from a stream buffer;

Stop: Stop decoding and displaying data from a stream buffer;

Get_Info: Get information about the current status of a device;

Set_Info: Set information in the device attributes.

The stream controller module 24 is responsible for the control of video and audio capture and playback functions during user-directed applications. This control includes maintaining the dynamic status of video and audio during capture or playback, and additionally, providing presentation control functions such as play, pause, step and reverse. This module is accordingly responsible for notifying an active application of stream events during audio and video capture or playback. An event is here defined as the current

presentation unit number, for which an indication would be made, or the occurrence of the matching of a prespecified presentation unit number with a current presentation unit number.

During the active playback of audio and video, or other dynamic media data streams, the stream I/O manager and the streams interpreter act as the time-based producer and consumer, respectively, of the data streams being played back. Conversely, during recording of a dynamic data stream, the stream interpreter acts as the time-based stream producer and the streams I/O manager acts as the time-based stream consumer. During both playback and recording, the I/O manager and the interpreter operate autonomously and asynchronously, and all data in an active stream flows directly between them via a well-defined data channel protocol. The stream controller asynchronously sends control messages to affect the flow of data between the I/O manager and the interpreter, but the controller does not itself participate in the flow of data. As discussed below, all data flow operations are handled using a minimal number of buffer copies between, for example, a disk or network subsystem and the digital video capture and presentation hardware.

This system design is particularly advantageous in that it provides for complete transparency with respect to the domain of the I/O manager and the interpreter, thereby providing the ability to extend the system to a network client/server configuration, as explained below. Moreover, this basic three-agent unit may be concatenated or recursed to form more complex data and control functionality graphs.

In the architecture of the local DVMS manager, the activity of one of the asynchronous agents, each time it is scheduled to run while participating in a stream flow, is represented as a process cycle. The rate at which an asynchronous agent is periodically scheduled is represented as the process rate for that agent, and is measured as process cycles per second. A process period is defined as the time period between process cycles. In order to maintain continuous data flow of streams between the stream I/O manager and the stream interpreter, the limiting agent of the two must process a process period's worth of presentation units within a given process cycle. In cases in which such process rates are not achieved, the local DVMS manager can control the flow rate, as explained below. The process rate for the stream interpreter is close to the nominal presentation rate of the stream, i.e., in every process cycle, a presentation unit is processed. The stream I/O manager services several presentation units in every process cycle and thus, its process rate may be much lower than the presentation rate.

The modularity of the stream control functions provided by the stream I/O manager, interpreter and controller make the local DVMS manager architecture of the DVMS highly portable to most modern computer operating systems which support preemptive multitasking and prioritized scheduling. This architecture also provides for selective off-loading of the stream I/O manager and interpreter modules to a dedicated coprocessor for efficient data management. Most importantly, the highly decentralized nature of the manager architecture allows it to be easily adapted to LAN and WAN systems, as discussed below.

Referring to FIG. **5**, when a computer implemented with the DVMS of the invention requests access to audio or video streams, the following stream flow occurs. The stream I/O manager **26** module retrieves the requested streams from a stream input **30**; this stream input comprises a storage access point, e.g., a computer file or analog video source. The stream I/O manager then separates the retrieved streams according to the specified file format of each stream. If two streams, e.g., audio and video streams, which are accessed were interleaved in storage, the stream I/O manager dynamically separates the streams to then transform them to distinct internal representations, each comprising a descriptor which is defined based on their type (i.e. audio or video). Once separated, the audio and video stream data are handled both by the stream I/O manager and the stream interpreter as distinct constituent streams within a stream group. The stream I/O manager **26** then exchanges the stream data, comprising sequences of presentation units, with the stream interpreter **28** via a separate queue of presentation units called a stream pipe **32**, for each constituent stream; an audio stream pipe **33** is thus created for the audio presentation units, and a video stream pipe **31** is created for the video presentation units. Each audio stream (of a group of audio streams) has its own pipe, and each video stream has its own pipe. During playback of streams, the stream I/O manager continually retrieves and produces presentation units from storage and the stream interpreter continuously consumes them, via the stream pipes, and delivers them to a digital media data subsystem for, e.g., presentation to a user.

When retrieving a plurality of streams from an input **30** in which the streams are separated (not interleaved), the stream I/O manager retrieves and queues the streams' data in a round robin fashion, but does not perform any stream separation function. The stream interpreter processes these streams in the same manner as it processes those which are originally interleaved. Thus, the stream I/O manager advantageously shields the remainder of the system from the nature of the static container **30**, and further "hides" the format of the storage container, as well as the way that logically coordinated data streams are aggregated for storage. Additionally, the details of the stream interpreter implementation, such as its hardware configuration, are "hidden" from the I/O subsystem; in fact, the only means of communication between the two agents is via the well-defined stream pipe data conduits.

Referring also to FIG. **6**, during a presentation scenario, the stream controller **24** first initializes **36** the stream I/O manager **26** and stream interpreter **28**, by creating active modules of them to begin processing streams, and then defines and indicates **38** a stream group and the corresponding constituent stream names. The stream I/O manager **26** then retrieves **40** the named streams from corresponding storage containers **30** and separates the streams, if stored in an interleaved fashion. If they were not interleaved, the streams are retrieved in a round-robin fashion. Once the streams are retrieved, the streams I/O manager converts **42** the streams to an internal computer representation of stream tokens, described below. Via the stream group indication **30**, each stream token is identified with a stream and a stream group by the indication provided to the stream I/O manager by the stream controller. The I/O manager then buffers **44** the streams separately, each in a distinct stream pipe **32** for consumption by the stream interpreter **28**; the stream controller provides control **46** of the steam group as it is enqueued.

Referring also to FIG. **7**, the I/O manager streams translation **42** from storage representation to stream token representation is as follows. Typically, audio and video data is stored in an interleaved fashion on a disk and so upon retrieval are in an interleaved disk buffer, as in the Intel® AVSS file format. The disk buffers **100** consist of a sequence of stream group frames **105**, each frame containing a header **106**, a video frame **108**, and an audio frame **110**. A separate index table (not shown) containing the starting addresses of

these stream group frames is maintained at the end of a file containing these frames. This index table permits random access to specifically identified stream group frames.

The disk buffers are retrieved by the I/O manager from the disk in large chunks of data, the size of each retrieved chunk being optimized to the disk track size, e.g., 64K bytes each. The I/O manager examines each retrieved stream group frame header and calculates the starting addresses of each audio and video frames within the stream group frame. It also retrieves the time stamp information from the corresponding frames. A linked list of descriptors, called tokens **112**, is then generated for the audio and video frames; each token represents an audio or video presentation unit **114** and the time stamp **116** for that unit. These tokens are continuously linked into a list representing the stream pipe. Thus, in the process described above, the stream I/O manager retrieves interleaved data from a disk, separates the data into distinct streams, and constructs an internal representation of separated streams based on separate stream pipes, one for each stream.

Once the streams are enqueued in the stream pipes, the stream interpreter **28**, having been initialized **36** by the stream controller **24**, accepts and dequeues **48** the constituent stream tokens of presentation units. The debuffered streams are then scaled **50** and synchronized **52**, based on control via the stream controller, which maintains **54** the status of the stream group. The scaling process will be described in detail below. The synchronized streams are then delivered to the digital presentation subsystem hardware.

The decompression scheme is based on the particular compression format of video frames, e.g., the motion JPEG video format. This format is one of a preferred class of video formats, in which each frame is intracoded, i.e., coded independently, without specification of other frames.

Referring to FIG. **8**, the digital video system **120** receives streams from the stream interpreter and first decodes and decompresses **122** the stream data, each stream being processed separately. The decoded and decorepressed data streams are then stored **124** in corresponding frame buffers, e.g., video and audio frame buffers. At the appropriate time, the stored data is converted **126** from its digital representation to a corresponding analog representation, and is delivered to a playback monitor and audio speakers. The various operations of the digital hardware subsystem are controlled by the stream interpreter via digital video primitives, as explained and described previously.

In the reverse operation, i.e., capture and storage of digital video and audio streams being processed by a computer system, the stream interpreter **28** captures the audio and video streams from the digital hardware subsystem **120**. Before this capture, the hardware subsystem digitizes **128** the audio and video signals, stores **130** the digitized signals in a buffer, and before passing the digitized streams to the stream interpreter, compresses and encodes **132** the video and audio data.

Based on the stream group control provided by the local stream controller, the stream interpreter generates **62** time stamps for the captured streams and using the time stamps, creates **64** corresponding stream tokens of video and audio presentation units with embedded time stamps. The stream tokens are then enqueued **66** to stream pipes **32** for consumption by the stream I/O manager **26**.

The piped streams are accepted and dequeued **72** by the stream I/O manager **26**, and then scaled. If the streams are to be stored in interleaved form, they are then interleaved **76**, in a process which reverses the functionality depicted in FIG. **7**. The streams are not required, of course, to be stored in such an interleaved form. Once the streams are interleaved, if necessary, the streams are stored in a corresponding storage container **30**.

Each of the functions of the stream controller, stream I/O manager, and stream interpreter described in these scenarios may be implemented in hardware or software, using standard design techniques, as will be recognized by those skilled in the art. Appendices A, B, and C present a pseudocode scheme for the interactions between the stream controller, stream I/O manager, and stream interpreter in retrieving and presenting streams. The coding of the pseudocode process steps into computer instructions suitable to carry out the described scenario will be understandable to one having ordinary skill in the art of C programming.

Synchronization of Audio with Video

As mentioned in the presentation process described above, the digital video management system of the invention provides synchronization of audio to video, and in general, synchronization between any two or more dynamic stream being presented. This synchronization function is inherently required for the coordinated presentation of multiple real-time, continuous, high data-rate streams in a stream group. For example, the real-time nature of audio and video is derived from the presentation attributes of these dynamic data types, which have quite different presentation attributes; full motion video needs to be presented as 30 frames per second and high quality audio needs to be presented at 32,000 samples per second.

Furthermore, digital video and audio data streams have real-time constraints with respect to their presentation. The streams are usually continuous and last from 30 seconds-long (clips) to 2 hours-long (movies). Additionally, the streams typically consume from about 1 Mbit/sec to 4 Mbit/sec of storage capacity and transmission bandwidth, depending on the particular compression technology used for digitizing the stream. Thus, synchronization of differing data streams must accommodate the diverse temporal aspects of the streams to be synchronized.

The synchronization capability of the digital video management system of the invention is based on self-timing, and accordingly, self-synchronization, of data streams to be synchronized. This technique accommodates independent handling of multiple data streams which are together constituent streams of a stream group, even if the stored representations of the constituent stream are interleaved; the stream I/O manager separates interleaved streams before the stream interpreter synchronizes the streams. Alternatively, independent constituent streams may, however, be stored in separate file containers and be synchronized, before presentation, with a common reference time base.

Self-synchronization also provides the ability to prioritize one constituent stream over other streams in a stream group. For example, an audio stream may be prioritized over a video stream, thereby providing for scalable video storage, distribution and presentation rates, as discussed below. This feature is particularly advantageous because human perception of audio is much more sensitive than that of video. For accurate human perception of audio, audio samples must be presented at a smooth and continuous rate. However, human visual perception is highly tolerant of video quality and frame rate variation; in fact, motion can be perceived even despite a wide variation in video quality and frame rate. Empirical evidence shows that humans can perceive motion if the presentation rate is between 15 and 30 frames/sec. At lower frame rates motion is still perceivable, but artifacts of previous motions are noticeable.

The DVMS of the invention exploits this phenomenon to optimally utilize available computing, compression and network resources; by prioritizing the retrieval, transmission, decompression and presentation of audio over video within a computer system or network computing environment, and by relying on audio-to-video synchronization before presentation, rather than at storage, an acceptable audio rate can be maintained while at the same time varying the video rate to accommodate resource availability in the system or network. Additionally, independent management of audio and video data streams provides many editing capabilities, e.g., the ability to dynamically dub a video stream with multiple audio language streams. Similarly, the synchronized presentation of an audio stream with still pictures is provided for by the independent stream management technique. It must be remembered that all of the synchronization schemes described are applicable to any type of stream, not just audio and video streams.

As described above with reference to FIG. 6, the synchronization of streams within a stream group is the responsibility of the stream interpreter module during a scaling process. The streams may be self-synchronized using either an implicit timing scheme or an explicit timing scheme. Implicit timing is based on the fixed periodicity of the presentation units in the constituent streams of a stream group to be synchronized. In this scheme, each presentation unit is assumed to be of a fixed duration and the presentation time corresponding to each presentation unit is derived relative to a reference presentation starting time. This reference starting time must be common to all of the constituent streams. Explicit timing is based on embedding of presentation time stamps and optionally, presentation duration stamps, within each of the constituent streams themselves and retrieving the stamps during translation of streams from the storage format to the token format. The embedded time stamps are then used explicitly for synchronization of the streams relative to a chosen reference time base.

Using either the implicit or explicit timing self-synchronization schemes, a reference time base is obtained from a reference clock, which advances at a rate termed the reference clock rate. This rate is determined by the reference dock period, which is the granularity of the reference clock ticks.

The DVMS of the invention supports two levels of self-synchronization control, namely, a base level and a flow control level. Base level synchronization is applicable to stream process scenarios in which the stream I/O manager is able to continuously feed stream data to the stream interpreter, without interruption, and in which each presentation unit is available before it is to be consumed. In this scenario, then, the stream I/O manager maintains a process rate and a process work load that guarantees that the stream I/O manager stays ahead of the stream interpreter.

The flow control level of synchronization is a modification of the base level scheme that provides a recovery mechanism from instantaneous occurrences of computational and I/O resource fluctuations which may result in the stream pipe between the stream I/O manager and the stream interpreter running dry. This could occur, for example, in a time-shared or multi-tasked computer environment, in which the stream I/O manager may occasionally fall behind the stream interpreter's demand for presentation units due to a contention, such as a resource or processor contention, with other tasks or with the stream interpreter itself. In such a scenario, the DVMS of the invention augments the base level of synchronization with a stream flow control function, as described below.

Base Level Implicit Timing Synchronization

As explained above, the base level synchronization scheme assumes that there is no need for control of stream flow to the stream interpreter, and thus does not monitor for vacancy of the stream pipe. Implicit timing is based on a reference time base that is applied to each stream to be synchronized.

Considering a scenario in which audio and video streams are to be synchronized, each presentation unit for the video stream to be presented might typically contain video information to be presented in a frame time of, e.g., 33 msec, for NTSC video play. The audio stream might typically be divided into fixed frames of presentation time with marginally varying samples per presentation unit. In a storage scheme in which the audio and video are interleaved, these fixed units of time are set as the time duration for a video frame, i.e., 33 msec.

In this synchronization scenario, the stream interpreter maintains a separate presentation unit counter for each stream pipe, and correspondingly, for each stream in the stream group. The interpreter consumes presentation units from the two streams in a round robin fashion, i.e., first one, then the other, and so on. Importantly, an independent presentation synchronization decision is made for each presentation trait, or token, of each stream, based on a corresponding reference time base, without regard to other streams. This reference time base indicates the current real time relative to the start time of the presentation unit consumption process for the corresponding stream. The stream counter of each stream pipe indicates the number of already consumed presentation units in the corresponding stream. Multiplying this count by the (fixed) duration of each of the presentation units specifies the real time which has elapsed to present the counted units. When this real time product matches the current reference time, the next presentation unit is released for presentation.

The stream interpreter initiates the consumption and presentation of each presentation unit in sequence during its presentation process cycle based on the presentation decision scheme given in pseudocode in Appendix D. This scheme implicitly assumes that the stream interpreter is scheduled such that the interpreter process rate is very close to the nominal presentation rate of the corresponding stream. This scheme is based on a comparison of a reference time base with the amount of time required to present the number of already-consumed presentation units, and thus requires the use of counters to keep a count of presentation units as they are consumed.

Base Level Explicit Timing Synchronization

As explained previously, in the explicit timing scheme, stream synchronization is based on time stamps that are embedded in the corresponding streams' tokens themselves. The time stomps represent the time, relative to the reference time base, at which the corresponding audio or video presentation frames are to be consumed and presented. The time base may be, for example, an external clock, or may be generated from the embedded time base of one of the streams to be synchronized. The periodicity of the time stamps is itself flexible and can be varied depending on particular synchronization requirements. Time stamps may be embedded in the streams during capture and compression operations, as described above, or at a later time during, for example, an editing process. Independent of the process by which the time stamps are embedded in a stream, the stamps are utilized by the stream I/O manager and interpreter during playback processes to make the consumption and presentation decisions. The stream interpreter does not maintain a

presentation unit counter in this scheme, as it does in the implicit timing scheme. Rather, the embedded time stamps in the streams provide equivalent information.

A time stamp for a presentation frame token consists of two 32-bit integers representing the presentation time and the presentation duration for that presentation unit. The presentation time and the presentation duration are represented in milliseconds. The presentation duration may be omitted if all presentation units are of the some duration.

In this synchronization scheme, the interpreter reads the embedded time stamp of each presentation token, as that token is processed, to determine presentation time and duration for each presentation unit in the sequence. The interpreter decides on consumption and presentation of each presentation unit in each stream based on the decision scheme given in pseudocode in Appendix E. This decision scheme is based on the assumption that the stream interpreter is scheduled such that its process rate is very close to the nominal presentation rate of the corresponding stream. This scheme is based on a comparison of a reference time base with the presentation time and presentation duration stamp embedded in each presentation unit. When a presentation unit's stamp presentation time corresponds to the reference time, that presentation unit is consumed for presentation.

In addition to determining the appropriate time for releasing presentation units in the sequence, both the implicit and explicit timing schemes delete presentation units if the appropriate release time for those units has passed. For example, in the implicit timing scheme, when the product of processed units and unit duration exceeds the currently maintained time count, the next sequential unit is deleted, rather than presented. Similarly, in the explicit timing scheme, then the current presentation time exceeds the time stamp presentation time of a presentation unit, that unit is deleted, rather than presented. In this way, synchronization of streams is maintained, even if units arrive for presentation at a later time than expected. The Appendices D and E give corresponding pseudocode for this presentation unit deletion function.

Flow Control Level Implicit Timing Synchronization

The flow control synchronization scheme augments the base level synchronization scheme to provide for recovery from instantaneous computational and I/O resource fluctuations during a consume and presentation process cycle. The base level scheme relied on the assumption that the stream I/O manager stays ahead of the stream interpreter to keep steam pipes from becoming vacant, or running dry. Flow control synchronization guards against this condition using a scheme based on virtual presentation units.

A virtual presentation unit is one which allows the underlying digital hardware subsystem to continue with a default presentation for the duration of a corresponding presentation unit, while at the same time maintaining a consistent internal state, to thereby provide sequential processing of a stream that is being presented, even while the stream pipe is temporarily empty. Virtual presentation units may be implemented in a variety of embodiments. For example, in the case of motion JPEG video, the playing of a virtual presentation unit would preferably correspond to redisplaying the most recent previous video frame. In the case of audio streams, a virtual presentation unit would preferably correspond to a null unit, i.e., a presentation unit consisting of null samples that represent silence. Other virtual presentation unit implementations are equally applicable.

During a presentation process cycle using the flow control implicit timing scheme to synchronize stream flow, the stream I/O manager and stream interpreter perform the same operations described above in the base level scheme. As explained, the interpreter maintains a separate presentation unit counter for each stream within the stream group being presented, to keep track of the number of already-consumed presentation units in each stream. Multiplying this count by the duration of each presentation unit specifies the time at which, when matching the reference time, the next presentation unit in the sequence is to be presented. The stream interpreter decides on the consumption and presentation of each presentation unit based on the decision scheme given in pseudocode in Appendix F, which assumes that the interpreter is scheduled at a process rate that is close to the nominal stream presentation rate. In this scheme, when the interpreter finds that a presentation token is not available from the stream pipe, and that the reference time and presentation unit count indicate that a presentation unit is needed, a virtual presentation unit is generated and consumed for presentation.

Flow Control Level Explicit Timing Synchronization

During a presentation process cycle using the explicit timing synchronization mechanism augmented with flow control capability, each presentation token in the stream group being presented is assumed to include its own embedded time stamp for presentation time and duration. As in the explicit timing scheme without flow control, the stream interpreter examines each embedded time stamp to decide on the consumption policy of the corresponding presentation unit in the stream pipes set up by the stream I/O manager. The consumption policy is determined based on the decision scheme, given in pseudocode in Appendix G, which assumes, as did the other schemes, that the process rate of the stream interpreter is close to the nominal presentation rate of the corresponding stream. In this scheme, when it is determined that another presentation unit is not available from the stream pipe and a unit should be presented, a virtual presentation unit is generated based on a default presentation duration, and that unit is then consumed for presentation.

Additionally, in the flow control schemes of either implicit or explicit timing, capability is provided to skip over presentation units. This capability is envoked whenever a previously unavailable presentation unit later becomes available. In the explicit timing scheme, the time stamp of a later available unit will never match the reference time after the presentation of a virtual presentation unit, and thus that unit will never be presented, and will be discarded. In the implicit timing scheme, the presentation of a virtual presentation unit in place of an unavailable presentation unit advances the presentation unit counter, as does any presented unit. When the unavailable unit is then later available, the presentation unit count will be advanced such that the product of the count and the fixed presentation unit duration will not permit presentation of that unit.

Coding of the four synchronization processes described above and in Appendices D-G into instructions suitable for implementing the synchronization techniques will be understandable to those having ordinary skill in the art of C programming.

Self-Synchronization Features

The four self-synchronization schemes described above provide several critical advantages in the digital video management scheme of the invention. Self-synchronization accommodates the ability to dynamically associate distinctly stored streams with a common stream group. Thus, for example, audio and video streams may be stored in separate file containers and grouped dynamically during retrieval from storage for synchronized presentation. As discussed

above, this synchronization of constituent audio and video streams provides, for example, for the function of dubbing of video with audio, and synchronizing still video with audio. Additionally, using the stream synchronization technique, stream segments from different file containers can be dynamically concatenated into one stream. In the case of explicit self-synchronization, the stream I/O manager marks the first presentation unit in a stream segment with a marker indicating the start of a new stream segment. Then when the stream interpreter consumes this presentation unit, the interpreter reinitializes the reference time base for the corresponding stream.

Self-synchronization further accommodates the ability to adapt to skews in the clock rates of audio and video hardware used to play audio and video streams which are being synchronized. For example, an audio stream recorded at an 11, 22 or 33 KHz sampling rate must be played back at exactly the sampling rate for accurate audio reproduction. Similarly, a video stream recorded at 30 frames per second must be played back at that same rate. The audio and video hardware playing these streams thus must each use clocks adapted for the particular play rate requirement of the corresponding stream. Any skew in the clock rates would cause drifting of the playing streams, and thus destroy synchronization of the streams, if the skew were to be uncorrected. Self-synchronization achieves this correction automatically using a reference time base which the audio and video time bases are checked against; the consumption rate of a stream is adjusted to drop presentation units periodically, if necessary, if a skew in one of the time bases, relative to its prescribed correspondence with the reference time base, is detected, thereby maintaining synchronization with respect to the reference time base and the other stream.

The self-synchronization schemes provide the capability to vary the inherent presentation rate of streams. For example, a video stream captured in PAL format, based on 25 frames per second, may be played in the NTSC format, which is 30 frames per second, albeit with some loss of fidelity. In general, any stream may be played at a custom rate, independent of the rate at which the stream was captured. In fact, it is often desirable in video playback to either speed up or slow down the nominal presentation rate of the video. Using the self-synchronization technique, the video presentation rate may be, for example, sped up by a factor of 2 by simply advancing the reference time base to twice the real time rate. Conversely, the presentation may be slowed by half by advancing the reference time base at one half the real time rate. In these cases, the total time elapsed for the presentation will be, of course, one half or twice the elapsed time for the presentation made at the nominal rate.

Stream Scalability

A scalable stream is a stream that can be played at an aggregate nominal presentation rate with variable data rates, under computer control. Of course, variation in the data rate may affect the quality, fidelity or presentation rate of the stream. The coupling of stream scalability with stream self-synchronization provides a powerful control mechanism for flexible presentation of audio and video stream groups.

As discussed above, scalability allows the DVMS to optimize utility of computer system resources by adjusting stream rates according to utility availability. In the case of audio and video streams, the stream interpreter may be programmed to give higher priority to audio streams than video streams, and thus consume audio presentation units at the nominal audio presentation rate, but consume video units at an available presentation rate. This available presentation rate is determined by the available computational resources of a given computer system. Different computer systems having varying performance characteristics require differing amounts of time to accomplish presentation operations. Such operations involve decompression, format conversion and output device mapping. In particular, a compressed Motion JPEG video stream has to be Huffman decoded, DCT decompressed, converted to RGB color space, and mapped to a 256 color VGA palette by the digital hardware subsystem before presentation within an IBM PC-compatible personal computer system; different computer systems require various time periods to accomplish these tasks. Thus, the management system of the invention adapts to any computer performance characteristics by adjusting the scale of the stream flow rate to accommodate the availability of utilities in that computer.

Most importantly, the stream scalability feature of the digital video management system of the invention provides the ability to comprehensively manage distribution of digital streams over packet networks. The DVMS exploits this capability in a network embodiment providing management protocol schemes for client-server sessions, as well as management protocol schemes for storing, accessing, retrieving and presenting streams over a LAN or WAN. The system thereby accommodates on-demand retrieval and playback of stored streams, and injection and tapping of multicast live streams over packet networks. The managed digital streams may be stored in ordinary computer files on file servers, or may be generated from live analog sources and made accessible over a LAN or WAN. Such access may be on-demand, as mentioned above, as in retrieval and presentation from a stored file, or on-schedule, as in injection and tapping from a broadcast channel. The management protocol schemes provided by the DVMS will be fully described below.

Referring now to FIG. 9, in a network implementation, the local DVMS manager 20 accesses digital media stream located elsewhere in the network via the remote DVMS manager 82 of the management system; the local DVMS manager provides a client operating environment, while the remote DVMS manager provides a network operating environment. Via the network 80, the local DVMS manager 20 and the remote DVMS manager 82 transmit control messages and digital media data streams as they are requested by a computer client connected in the network.

Remote DVMS Manager

The remote DVMS manager 82 manages network control of digital media streams via four independent modules, namely, a remote stream controller 84, a remote stream input/output (I/O) manager 86, a remote network stream I/O manager 88, and a local network stream I/O manager 90.

In this DVMS network implementation, the local DVMS manager 20, residing locally to a client computer in the network, comprises a local stream controller 24, local stream I/O manager 26 and local stream interpreter 28. The local network stream I/O manager 90 of the remote DVMS manager directly interfaces with the local DVMS manager locally.

The remote stream controller 84 resides on a remote storage device or access point, e.g., a video server, in the network. This controller is responsible for managing the remotely stored streams, e.g., video files, and thereby making them available for on-demand access by the local stream controller module of the local DVMS manager. Client-server session management protocols control this access. The remote stream controller also provides a link for feedback control from the local DVMS manager to the remote DVMS manager, as described below.

The remote stream I/O manager **86** also resides on a remote server; it is responsible for dynamically retrieving and storing streams from or to a storage container in the remote storage server. Efficient access to stored stream information and handling of file formats is provided by this module. Thus, the remote stream I/O manager performs the same tasks as those performed by the steam I/O manager of the local DVMS manager in a stand-alone computer implementation—tasks including translation between stored stream representations and corresponding dynamic computer-based token representations.

The remote network stream I/O manager **88**, implemented on a remote server, regulates transmission of streams across the network to and from a local DVMS manager with which a communications session has been initiated. This transmission comprises stream exchange between the remote network stream I/O manager **88** and the local network stream I/O manager **90**, which resides locally with respect to the local DVMS manager modules, on a client in the network. Stream transport protocols control the transmissions. The local network stream I/O manager **90** receives streams from the network and delivers them to the local DVMS stream interpreter **28** during playback processes; conversely, it receives streams from the local stream interpreter and transmits them over the network during recording and storage processes.

The DVMS of the invention provides protocols for managing the interaction and initialization of the local DVMS manager modules and the remote DVMS manager modules just described. Specifically, four classes of protocols are provided, namely, access protocols, for stream group naming and access from a stream server or injector; transport protocols, providing for stream read-ahead, and separation and prioritization of streams; injection/tap protocols, providing the capability to broadcast scheduled streams, e.g., video streams, to selected network clients; and feedback protocols, accommodating the management of adaptive computational resources and communication bandwidths.

When the DVMS is configured in a network environment, remote media data stream file servers in the network advertise the stream groups controlled in their domain based on a standard network advertisement protocol. For example, in the Novell® Netware™ environment, servers advertise based on the Service Advertisement Protocol (SAP). Each video server is responsible for a name space of stream group containers that it advertises.

As shown in FIG. **9**, when an application running on a computer (client) connected in the network opens a stream group container by name to access the container contents, the DVMS initializes the corresponding local stream controller **24** of the local DVMS manager to access the corresponding stream group. The local stream controller then sets up a client-server session with the appropriate remote stream controller **82** based on the stream group container name that the application wishes to access and the remote server's advertisement. The local stream controller may access multiple stream group containers during a single session. This capability results from the name service architecture employed by the remote DVMS manager. In this scheme, a domain of container names is accessed via a single access call, whereby multiple containers in the domain are simultaneously available for access.

The local stream controller **24** then initializes the local network stream I/O manager **90** of the remote DVMS manager, and commences a stream read-ahead operation, described below, with the appropriate remote stream controller **84**. In turn, that remote stream controller initializes the corresponding remote stream I/O manager **86** and remote network stream I/O manager **88** to handle retrieval and transmission of the constituent streams within the accessed stream group.

The stream read ahead operation is employed to reduce latency perceived by a client when a stream group presentation is begun; stream retrieval, transmission, and scaling require a finite amount of time and would be perceived by a client as a delay. In the read ahead operation, the remote stream I/O manager, the remote network stream I/O manager, and the local network stream I/O manager retrieve, transmit, and scale the streams at the very start of a client-server session, even before the client requests stream presentation. In this scheme, the stream are ready for immediate consumption by the local stream interpreter, via the stream pipes, whenever a user specifies the start of presentation, and possible presentation delays are thereby eliminated or minimized.

Referring now to FIG. **10**, when a network client requests access to a specified stream group, the following procedure is implemented. Upon initialization from the request, and based on the network servers' stream group advertisements, the appropriate remote stream I/O manager **86** retrieves stored streams, e.g., audio and video streams, from the appropriate file storage **30** containing the requested stream group. The manager then separates the retrieved streams, if necessary, thereby producing separate audio and video presentation unit streams, and enqueues corresponding stream descriptor tokens in separate stream pipes **87**, one pipe for each presentation unit token stream.

The remote network stream I/O manager **88** consumes the presentation unit tokens from each of the stream pipes, assembles transmission packets based on the streams, and releases them for transmission across the network **80** directly to the corresponding local network stream I/O manager **90**, based on the DVMS stream data transport protocols; the particular transport protocol used is set by the network environment. For example, in a Novell® network, the Netware SPX protocol is used for stream data transport. The local network stream I/O manager **90**, upon receipt of the transmitted presentation units, queues the presentation units in separate stream pipes **32** for each stream to be consumed by the local stream interpreter **28** for use by the client computer's digital media hardware subsystem **34**.

Referring to FIG. **11A**, illustrating the remote DVMS functions in more detail, upon initialization, the remote stream controller **84** initializes the remote stream I/O manager **86** and the remote network stream I/O manager **88** by creating **130**, **136** active modules of each of the managers. It also specifies **132** the requested stream group for access by the two managers. Control **134** of the specified stream group is provided throughout the duration of the managers' functions.

The remote stream controller **84** also provides management **138** of the client/server session which proceeds between the local and remote DVMS systems as a result of the stream group request. Based on information provided by the local DVMS manager which requested the stream group, the remote stream controller receives **140** a desired rate value from the local DVMS; this rate value indicates the rate at which the streams are to be presented, and is explained more fully below. The remote stream controller specifies **142** this rate to each of the remote stream I/O manager **86** and the remote network stream I/O manager **88**, which each receive **144** the rate.

The remote stream I/O manager **86** retrieves, separates, and scales **146** audio and video streams from the appropriate

stream container 30. If the streams were stored separately, rather than interleaved, the streams may be individually scaled at this point, while if the streams were interleaved, the remote network stream I/O manager 88 later scales the streams, as explained in detail below.

In a process explained previously with reference to FIG. 7, the remote stream I/O manager creates 148 stream tokens corresponding to the stream presentation unit frames retrieved from storage, and enqueues 150 the stream tokens for delivery to the remote network stream I/O manager via individual stream pipes 32.

The remote network stream I/O manager 88 dequeues 152 the tokens from the stream pipes and if necessary, scales 154 the tokens. The tokens are then formatted 156 for transmission packets, and released to the network for transmission.

Referring also to FIG. 12, the packet format process 156 is implemented as follows. Each token 114 in the token streams 112 is enqueued in a buffer 118, whereby each buffer contains tokens and associated media frame data from one stream only, even if the streams were originally interleaved in storage. Tokens, along with corresponding media data from the buffers, are then sequentially ordered in packets 120 in such a manner that each token and the corresponding media data remain associated. This association, along with the fact that tokens are likely to be time stamped, does not require that the storage format and congruency of the stream be preserved in the transmission packets during transmission.

This packet format scheme provides dramatic advantages over the conventional packet format scheme of the prior art. In the conventional packet protocol the stored media data format, which is typically interleaved, is preserved in the transmission packet format. Thus, in this scheme, audio and video streams are transmitted across a network in packets containing a sequence of interleaved headers, audio frames, and video frames, and thus, the specific syntax by which the interleaved streams were stored is replicated in the packet format.

In contrast, in the packet format scheme of the invention, the separation of streams and distinctly formatting of packets for each stream provides an opportunity and the facility to examine, process, and make transmission decisions about each stream and each presentation unit independent of other streams and presentation units. As a result, the local DVMS manager can make presentation decisions about a given presentation unit token independent of the other tokens in the corresponding stream, and can make those decisions "on-the-fly". This capability provides for real time scaling and network load adjustment as a stream is retrieved, processed, and transmitted across the network. The conventional prior art scheme does not have any analogous facility, and thus cannot provide the synchronization, scaling, and rate control features of the invention.

Referring to FIG. 11B, once the stream group is transmitted across the network, the local DVMS manager processes the stream group for presentation. The local stream controller 24 manages 158 the client/server session communication with the remote stream controller 84. Like the remote stream controller, it also creates 160, 162 instances of active processors, here initializing the local network stream I/O manager 90 and the local stream interpreter 28. The local stream controller creates 164 the stream grouping of interest and controls 166 that group as the local network stream I/O manager 90 and stream interpreter 28 process the group.

The local network stream I/O manager 90 receives 168 the transmitted network packets and assembles presentation units as they are received. Then it creates 170 stream tokens from the received packets and enqueues 172 them to individual stream pipes. The stream interpreter 28 dequeues 176 the tokens from the stream pipes and scales 176 the tokens as required, in a process discussed below. Then using the synchronization schemes explained previously, the streams are synchronized 178 and sent to the digital hardware subsystem for presentation. The functions of this hardware were explained previously with reference to FIG. 8.

In the reverse process, i.e., when recording streams from a network client for storage on a remote stream server, as shown in FIGS. 11A and 11B, the digital stream hardware subsystem provides to the local stream interpreter 28 the stream data, and based on the playing format of the streams, the local stream interpreter generates 180 corresponding time stamps, for use in synchronization and scaling. Stream tokens are then created 182 and enqueued 184 in the stream pipes.

The local network stream I/O manager dequeues 186 the stream tokens from the pipes and scales 188 the streams based on their play rate, record rate, and storage format, as discussed below. Then packets are formed and transmitted 190 via the network to the remote server location on which the corresponding remote DVMS exists.

Thereafter, the remote network stream I/O manager 88 receives 192 the transmitted packets and creates 194 stream tokens based on the packets. The tokens are then enqueued 196 in stream pipes for consumption by the remote stream I/O manager. The remote stream I/O manager dequeues 198 the tokens from the stream pipes, and scales 200 the streams if necessary. Finally, it interleaves the streams, if they are to be stored in an interleaved format, and stores 202 the streams in appropriate stream containers on the server.

FIGS. 11A and 11B illustrate that the network implementation of the DVMS of the invention is an elegant and efficient extension of the stand-alone DVMS implementation; this extension is possible as a result of the modularity in design of each processing entity. Specifically, the details of packet transport are transparent to the remote stream I/O manager; it functions in the some manner as a stand-alone stream I/O manager. Similarly, presentation unit token streams provided to the local stream interpreter do not contain transmission-specific formats.

As a result, the local DVMS manager, when implemented in a network environment, is easily reconfigured to provide a remote DVMS manager which includes a corresponding remote stream I/O manager, with the addition of a remote network stream I/O manager; and a local DVMS manager which includes a corresponding local stream interpreter, and a local network stream I/O manager from the remote DVMS manager. Exploiting this modularity, programming applications may be created which are supported by the DVMS functionality without them perceiving a functional difference between a local, stand-alone type stream scenario and a remote, network stream scenario.

Appendices H, I, J, and K together present a C-language pseudocode implementation of the client-server session control and remote and local stream processing techniques required in addition to those given in Appendices A, B, and C for the network implementation of the DVMS of the invention. Those having ordinary skill in the art of C programming will understand the coding of theses pseudocode processes into corresponding code. Additionally, as will be recognized by those skilled in the art, these processes may alternatively be implemented in hardware using standard design techniques to provide the identical functionality.

Scalable Stream Rate Control

In the network embodiment of the DVMS of the invention, the remote and local DVMS managers operate together to provide control of the rate of flow of streams through a network during stream transmission. As mentioned above, this capability is particularly advantageous in handling audio and video streams to accommodate fluctuations in network utility availability by prioritizing audio stream rate over video stream rate.

This priority is based on the premise that human visual perception of motion is highly tolerant of variations in the displayed quality and frame rate of presented video. Typically, humans perceive motion when a video presentation rate exceeds at least 15 frames per second. Moreover, instantaneous and smooth variations in video presentation rates are practically unnoticeable. However, human aural perception is quite intolerant of variations in audio presentation quality or rate. Typically, humans perceive noise when a constant audio presentation rate is not maintained, and perceive "clicks" when brief periods of silence are injected into an audio stream. Thus, the DVMS system prioritizes audio streams over video streams. This prioritization of audio over video extends over the entire data flow of audio and video streams in a network, starting from their retrieval from storage containers and ending with their presentation.

Control of the rate of streams through a network based on this audio prioritization scheme may be initiated automatically, or in response to a direct user request. Each type of control request is discussed below in turn. The remote DVMS manager responds to each type in the same manner, however.

Referring again to FIG. 11A, remote stream controllers **84** in the network are responsible for instructing the corresponding remote stream I/O manager **86** and remote network stream I/O manager **88** as to what percentage of the nominal presentation rate (at which the stream would "normally" be presented) the stream should be actually retrieved and transmitted. The remote stream controller receives **140** the desired rate value via network communication with the local stream controller **24** and specifies **142** this rate to the remote stream I/O manager **86** and the remote network stream I/O manager **88**, which each receive **144** the rate value.

The stream rate control mechanism is carried out by either the remote stream I/O manager or the remote network stream I/O manager, depending on particular stream success scenarios. As explained above. If the requested audio and video streams are interleaved in storage, in, e.g., the Intel DVI AVSS file format, the remote stream I/O manager retrieves the streams in that interleaved form, separates the streams into distinct streams, and creates corresponding presentation unit tokens. The remote stream I/O manager does not, in this scenario, have the ability to manipulate the streams distinctly because they are retrieved interleaved. In this case, the remote network stream I/O manager, which obtains the streams from the stream pipe after they have been separated, controls the rate of each stream as before forming stream packets for network transmission.

If the streams to be retrieved are individually stored, the remote stream I/O manager may control the rate of the streams as they are each separately retrieved and corresponding tokens are created. In this case, the rate control functionality of the remote network stream I/O manager is redundant and does not further change the stream rate before the stream is transmitted across the network.

Rate control of noninterleaved streams is provided by the remote stream I/O manager during the scaling process **146**, in which case the remote stream I/O manager retrieves stream frames from the storage container while skipping over appropriate stream frames to achieve the prespecified stream rate. The streams frames which are skipped over are determined based on the particular compression technology that was applied to the stream. The remote stream I/O manager substitutes virtual presentation units for the skipped stream frames to maintain sequential continuity of the stream.

As explained previously regarding flow control synchronization schemes, a virtual presentation unit comprises a presentation unit with some amount of substitute media data information for maintaining a consistent internal state of stream unit sequence, even while a next sequential unit is unavailable. Here in the case of scaling, where virtual units are employed to scale the transmission rate of streams, virtual units are additionally employed to reduce the amount of presentation unit data that is transmitted.

Accordingly, here a virtual video presentation unit comprises a null presentation unit, having a specified presentation duration and time, or a time stamp, but not containing any frame presentation information. Then, when the remote stream I/O manager substitutes a virtual presentation unit for a skipped stream frame, a transmission packet including the virtual presentation unit is shorter and more quickly transmitted than it would be if the skipped frame was included. When the local stream interpreter and digital presentation subsystem receive and process the null video unit, they interpret that unit as an instruction to represent the most recently presented frame. In this way, the presentation subsystem maintains default video presentation data without requiring that data to be received via a network transmission.

As will be recognized by those skilled in the art of compression technology, it is alternatively possible, using appropriate compression techniques, to substitute partial media information, rather than null information to increase or decrease the transmission rate of presentation streams containing presentation units that will not be presented.

Rate control of interleaved stream is provided by the remote network stream I/O manager upon receipt of the stream tokens form the stream pipes. Here, the remote network stream I/O manager scales **154** the stream tokens as they are processed to form transmittal packets. This is accomplished by processing the stream in a scheme whereby the remote network stream I/O manager skips over appropriate tokens and substitutes virtual presentation unit tokens in their place, depending on the compression technology used, to achieve the specified stream rate.

In this common and important situation of interleaved stream storage, the remote network stream I/O manager participates in stream data flow and thus may be characterized with a particular process cycle and process period. During each of its process cycles, the remote network stream I/O manager processes a single presentation unit and determines if the next sequential presentation unit is to be transmitted based on a transmit decision scheme. Like the process decision schemes described above in connection with synchronization techniques, the transmit decision scheme is implemented based on the timing technique of the stream being processed; if the stream presentation units include embedded time stamps, then the transmit decision scheme is based on an explicit timing count, while implicit timing counting is employed otherwise.

No matter which agent provides the scaling function, only video streams are scaled, while audio stream presentation frames and tokens are processed at the full nominal presentation rate, without skipping any audio presentation frames; this preservation of audio presentation rate inherently prioritizes audio streams over video streams.

The scaling function is, as explained above, dependent on the compression technology employed for a particular frame or stream group. Using, e.g., a key frame-based compression technique, a key frame is an independently selectable frame within a stream that contains information required for decompression of all the following non-key frames dependent on that key frame. Dependent, or non-key, frames are not independently selectable. The motion JPEG format relies on a scheme in which every frame in a stream is a key frame. During the scaling operation, only key frames are skipped over, whereby all non-key frames associated with the skipped key frame are also skipped over. Null frames are then substituted for the key frame and all of its corresponding non-key frames.

Appendices L and M provide C-language pseudocode implementing an implicit timing rate control scheme and an explicit timing rate control scheme. Like the synchronization techniques described previously, the implicit rate control scheme is based on a counting technique and does not require embedded time codes on the stream presentation frames. The explicit rate control scheme is based on the use of time stamps for explicitly determining the presentation and duration time of a given frame. In either implementation, virtual presentation units are generated to accommodate skipped stream frames.

In addition, in either implementation, when skipped stream frames later become available, they are identified and skipped over, thereby being deleted, rather than presented. This presentation unit deletion function, like that employed in the synchronization schemes, maintains a current sequential stream progression. Appendices L and M provide pseudocode for implementing this presentation unit deletion function.

Adaptive Load Balancing

The DVMS of the invention includes the ability to automatically and dynamically sense the load of a packet network in which the system is implemented. Based on the sensed loading, the stream rate control mechanism described above is employed by the system to correspondingly and adaptively balance the load within the network, thereby optimizing the network utility availability.

Referring to FIG. 11B, in the this load balancing scheme, the local network stream I/O manager **90** monitors **206** the stream pipes **32** currently transmitting streams between that manager and the local stream interpreter **28** for variations in the average queue size, i.e., availability of presentation unit tokens, of each pipe. When the average queue size varies significantly, the local network stream I/O manager detects the direction of the change, i.e., larger or smaller. Thereafter, it notifies **208** the local stream controller **24** of the change and requests a new stream presentation token rate to be transmitted as a percentage of the nominal presentation rate, based on the change. In turn, the local stream controller transmits the request to the remote stream controller **84**, which in response, instructs the remote stream I/O manager **86** and the remote network stream I/O manager **88** to adjust the stream presentation unit rate to the requested rate.

The requested rate is based on the average queue size in the following scheme. When the queue size increases significantly above a prespecified upper availability, the requested rate is increased; the increased availability indicates that high-speed processing may be accommodated. Conversely, when the queue size decreases significantly below a prespecified lower availability, the requested rate is decreased; the decreased availability indicates that the current rate cannot be accommodated and that a lower rate is preferable.

Alternatively, a user may specify a desired stream presentation rate, that specification being accepted **204** by the local stream controller **24**. In turn, the local stream controller sends the request to the remote stream controller for implementation.

In the corresponding reverse process, in which stream frames are stored after being recorded via the local DVMS manager, the remote stream I/O manager scales **200** the stream before storage to reconstruct the stream such that it no longer includes null frames. This function may also be accomplished by the local network stream I/O manager in a scaling process **188** completed before a stream is transmitted.

The DVMS of the invention has been described with particular detail relating to a preferred embodiment. Other embodiments are intended to fall within the scope of the invention. For example, while the DVMS of the invention has been described in a scheme for managing audio and video streams, other media data stream types, e.g., stills, accessed from various media data access points, e.g., a PBX server, are within the scope of the claims. If the DVMS is implemented on a computer system or network in software, programming languages other than the C programming language may be employed, as will be clear to those skilled in the art of programming. Alternatively, the DVMS may be implemented entirely in hardware using standard digital design techniques, as will also be clear to those skilled in the art of digital hardware design.

Appendix A

Local Stream Controller

```
Local_Stream_Controller ( ... ) {
CNTRLR_MSG message;     /* Stream Controller Message structure */
    initialize ( ... );
    while ( for_ever ) {
        message = receive_message ( ... );
        switch ( message.operation ) {
            case OPEN: /* Open a Stream Group Player instance */
                ...
                hStream_Interpreter = Create_Stream_Interpreter ( ... );
                hLocal_Stream_IO_Manager = Create_Local_Stream_IO_Manager ( ... );
                hLocal_Network_Stream_IO_Manager =
                    Create_Local_Network_Stream_IO_Manager ( ... );
                ...
                break;
            case CLOSE:    /* Close a Stream Group Player instance */
                ...
                Delete_Stream_Interpreter ( hStream_Interpreter );
                Delete_Local_Stream_IO_Manager ( hLocal_Stream_IO_Manager );
                Delete_Local_Network_Stream_IO_Manager ( hLocal_Network_Stream_IO_Manager );
                ...
                break;
            case LOAD:/* Load a Stream Group by name */
                ...
                hStream_Group = Create_Stream_Group ( sStream_Group_Container, ... );
                if ( local ( hStream_Group )) {
                    send_message ( hLocal_Stream_IO_Manager, LOAD, hStream_Group, ... );
                }
                else {
                    /* Find and connect to the Remote Stream Controller */
                    hRemote_Stream_Controller = find (hStream_Group, ... );
                    connect (hRemote_Stream_Controller, ... );
                    /* Open a remote Stream Group player instance */
                    send_message ( hRemote_Stream_Controller, OPEN, ... );
```

```
                /* Initiate a remote loading of the Steram Group and
                    obtain a handle to the Stream Transport Channel */
                send_message ( hRemote_Stream_Controller, LOAD, hStream_Group,
                                phStream_Channel, ... );
                /* Pass the handle to the Stream Transport Channel to the Local Network Stream I/O
                    Manager and fill the Stream Pipes from the network */
                send_message ( hLocal_Network_Stream_IO_Manager, LOAD, hStream_Group,
                                *phStream_Channel, ... );
            }
            ...
            break;
        case UNLOAD: /* Unload a Stream Group by handle*/
            ...
            break;
        case PLAY: /* Play forward the loaded Stream Group */
            ...
            if ( local ( hStream_Group )) {
                send_message ( hLocal_Stream_IO_Manager, PLAY, hStream_Group, ... );
            }
            else {
                send_message ( hRemote_Stream_Controller, PLAY, hStream_Group, ... );
                send_message ( hLocal_Network_Stream_IO_Manager, PLAY, hStream_Group, ... );
            }
            send_message (hStream_Interpreter, PLAY, ... );
            ...
            break;
        case STOP: /* Stop and rewind the loaded Stream Group */
            ...
            break;
        case PAUSE:    /* Pause the playing Stream Group */
            ...
            break;
        ...
    }
}
} /* End Local_Stream_Controller ( ) */
```

A2

Appendix B

Stream I/O Manager

```
Stream_IO_Manager ( ... ) {
Int  nStreams;   /* Number of independent Streams in a Stream Group */
IOMGR_MSG  message;    /* Stream I/O Manager Message structure */
    initialize ( ... );
    while ( for_ever ) {
        message = receive_message ( ... );
        switch ( message.operation ) {
            case LOAD: /* Load a Stream Group and fill the Stream Pipes */
                state = LOADED;
                /* set number of independent streams in the Stream_Group*/
                nSreams = message.hStream_Group.nStreams;
                ...
                break;
            case UNLOAD:  /* Unload a Stream Group and clear the Stream Pipes */
                state = UNLOADED;
                ...
                break;
            case PLAY: /* Start retrieving data and feeding the Stream Pipes */
                state = PLAYING;
                ...
                break;
            case PAUSE:    /* Stop retrieving data and feeding the Stream Pipes */
                state = LOADED;
                ...
                break;
            ...
            ...
        }
        if ( state == PLAYING ) {
            int  i;
            ...
            for ( i = 0; i <= nStreams; i++ ) {
                ...
```

```
            Retrieve ( next presentation unit );
            Enqueue ( next presentation unit );
            ...
        }
        ...
    }
}
} /* End Stream_IO_Manager ( ) */
```

B2

Appendix C

Stream Interpreter

```
Stream_Interpreter ( ) {
int  nStreams;   /* Number of independent Streams in a Stream Group */
INTRPRTR_MSG   message;    /* Stream Interpreter Message structure */
    initialize ( ... );
    while ( for_ever ) {
        message = receive_message ( ... );
        switch ( message.operation ) {
            case LOAD: /* set number of independent streams in the Stream_Group*/
                nSreams = message.hStream_Group.nStreams;
                ...
                break;
            case UNLOAD:  /* */
                ...
                break;
            case PLAY: /* */
                ...
                break;
            case PAUSE:    /* */
                ...
                break;
            ...
            ...
        }
        if ( state == PLAYING ) {
            int  i;
            ...
            for ( i = 0; i <= nStreams; i++ ) {
                ...
                Present ( ... );    /* Present the next presentation unit */
                ...
            }
            ...
        }
```

```
    }
}   /* End Stream_Interpreter ( ) */
```

Appendix D

Base Level Implicit Timing Synchronization

```
#define T   < fixed presentation duration of a presentation unit >
int     p;  /* consumed presentation units */
int     t;  /* reference time base */
Present ( ... ) {
boolean done = FALSE;
if (t < p*T) { /* Continue presenting current presentation unit */
    return;
}
while (!done) {
    /* Consume and play a new presentation unit */
    if ((p*T <= t) && (t < (p+1)*T)) {
        Consume_and_Present[1] ( next presentation unit );
        p = p + 1;
        done = TRUE;
    }
    /* Catch up to current time relative to reference time base */
    if ((p+1)*T <= t) {
        Consume_and_Process[2]( next presentation unit );
        p = p + 1;
    }
}
}   /* End Present ( ) */
```

[1]Consume and Present operation refers to any decompression and processing required for presentation.
[2]Consume and Process operation includes decompression and internal state maintenance for algorithms using temporal prediction.

Appendix E

Base Level Explicit Timing Synchronization

```
#define T   < fixed presentation duration of a presentation unit >
int     p;  /* presentation time of next presentation unit */
int     d;  /* presentation duration of next presentation unit */
int     t;  /* reference time base */
Present ( ... ) {
boolean done = FALSE;
if (t < p + d) {
    /* Continue presenting current presentation unit */
    return;
}
while (!done) {
    /* Get new presentation time and duration */
    p = presentation_time ( next presentation unit );
    d = presentation_duration ( next presentation unit );
    /* Consume and play a new presentation unit */
    if ((p <= t) && (t < (p+d))){
        Consume_and_Present ( next presentation unit );
        done = TRUE;
    }
    if ((p + d) <= t) { /* Catch up to current time relative to reference time base */
        Consume_and_Process ( next presentation unit );
    }
}
}   /* End Present ( ) */
```

Appendix F

Flow Control Implicit Timing Synchronization

```
#define T   < fixed presentation duration of a presentation unit >
int     p;  /* consumed presentation units */
int     t;  /* reference time base */
int     vpu_count;  /* differential count of virtual presentation units */
Present ( ... ) {
boolean done = FALSE;
while (vpu_count) {  /* Consume and drop redundant presentation units */
    Consume_and_Process ( next presentation unit );
    vpu_count--;
}
if (t < p*T) { /* Continue presenting current presentation unit */
    return;
}
while (!done) {
    if (Stream_Pipe != EMPTY) {
        if ((p*T <= t) && (t < (p+1)*T)) {
            /* Consume and play a new presentation unit */
            Consume_and_Present ( next presentation unit );
            p = p + 1;
            done = TRUE;
        }
        /* Catch up to current time relative to reference time base */
        if ((p+1)*T <= t) {
            Consume_and_Process ( next presentation unit );
            p = p + 1;
        }
    else {
        if ((p*T <= t) && (t < (p+1)*T)) {
            /* Generate and play a new presentation unit */
            Fabricate( virtual presentation unit );
            vpu_count++;
            Consume_and_Present ( virtual presentation unit );
            done = TRUE;
```

```
            p = p + 1;
        }
    }
}
} /* End Present ( ) */
```

F2

Appendix G

Flow Control Explicit Timing Synchronization

```
#define D   < default presentation duration of a virtual presentation unit >
int     p;  /* presentation time of next presentation unit */
int     d;  /* presentation duration of next presentation unit */
int     t;  /* reference time base */
Present ( ... ) {
boolean done = FALSE;
if (t < p + d) {
    /* Continue presenting current presentation unit */
    return;
}
while (!done) {
    if (Stream_Pipe != EMPTY) {
        /* Get new presentation time and duration */
        p = presentation_time ( next presentation unit );
        d = presentation_duration ( next presentation unit );
        if ((p <= t) && (t < (p+d))){ /* Consume and play a new presentation unit */
            Consume_and_Present ( next presentation unit );
            done = TRUE;
        }
        if ((p + d) <= t) { /* Catch up to current time relative to reference time base */
            Consume_and_Process ( next presentation unit );
        }
    } else   {
        p = p + d;
        d = D;
        if ((p <= t) && (t < (p+d))){
            Fabricate( virtual presentation unit );
            Consume_and_Present ( virtual presentation unit );
            done = TRUE;
        }
    }
    }
}   /* End Present ( ) */
```

Appendix H

Remote Stream Controller

```
Remote_Stream_Controller ( ... ) {
int nStreams;   /* Number of independent Streams in a Stream Group */
CNTRLR_MSG message;   /* Stream Controller Message structure */
    initialize ( ... );
    while ( for_ever ) {
        message = receive_message ( ... );
        switch ( message.operation ) {
            case OPEN: /* Open a Stream Group Player instance */
                ...
                hRemote_Stream_IO_Manager = Create_Remote_Stream_IO_Manager ( ... );
                hRemote_Network_Stream_IO_Manager =
                    Create_Remote_Network_Stream_IO_Manager ( ... );
                ...
                break;
            case CLOSE:   /* Close a Stream Group Player instance */
                ...
                Delete_Remote_Stream_IO_Manager ( hRemote_Stream_IO_Manager );
                Delete_Remote_Network_Stream_IO_Manager
                        ( hRemote_Network_Stream_IO_Manager );
                ...
                break;
            case LOAD: /* Load a Stream Group by name */
                ...
                hStream_Group = Create_Stream_Group ( sStream_Group_Container, ... );
                send_message ( hRemote_Stream_IO_Manager, LOAD, hStream_Group, ... );
                /* Obtain a handle to the stream channel from the Remote Network Stream I/O Manager */
                send_message ( hRemote_Network_Stream_IO_Manager, LOAD, hStream_Group,
                                phStream_Channel, ... );
                /* Reply to the Local Stream Controller and return a handle to the stream channel */
                *message.phStream_Channel = *phStream_Channel
                reply_message ( message.sender, ... );
                ...
                break;
```

```
            case UNLOAD: /* Unload a Stream Group by handle*/
                ...
                break;
            case PLAY: /* Play forward the loaded Stream Group */
                ...
                send_message (hRemote_Stream_IO_Manager, PLAY, ... );
                send_message (hRemote_Network_Stream_IO_Manager, PLAY, ... );
                ...
                break;
            case STOP: /* Stop and rewind the loaded Stream Group */
                ...
                break;
            case PAUSE:    /* Pause the playing Stream Group */
                ...
                break;
            ...
        }
    }
}  /* Remote_Stream_Controller ( ) */
```

H2

Appendix I

Remote Stream I/O Manager

```
Remote_Stream_IO_Manager ( ... ) {
int  nStreams;   /* Number of independent Streams in a Stream Group */
IOMGR_MSG  message;    /* Stream I/O Manager Message structure */
    initialize ( ... );
    while ( for_ever ) {
        message = receive_message ( ... );
        switch ( message.operation ) {
            case LOAD: /* Load a Stream Group and fill the Stream Pipes */
                state = LOADED;
                /* set number of independent Streams in the Stream Group*/
                nSreams = message.hStream_Group.nStreams;
                ...
                break;
            case UNLOAD:  /* Unload a Stream Group and clear the Stream Pipes */
                state = UNLOADED;
                ...
                break;
            case PLAY: /* Start retrieving data and feeding the Stream Pipes */
                state = PLAYING;
                ...
                break;
            case PAUSE:    /* Stop retrieving data and feeding the Stream Pipes */
                state = LOADED;
                ...
                break;
            ...
            ...
        }
        if ( state == PLAYING ) {
            int  i;
            ...
            for ( i = 0; i <= nStreams; i++ ) {
                ...
```

```
            Enqueue ( next presentation unit );
            ...
          }
          ...
        }
      }
    }
  /* End Remote_Stream_IO_Manager */
```

Appendix J

Remote Network Stream I/O Manager

```
Remote_Network_Stream_IO_Manager ( ... ) {
Int nStreams;   /* Number of independent Streams in a Stream Group */
IOMGR_MSG message;    /* Stream I/O Manager Message structure */
Int hStream_Channel;   /* Handle to the Stream Transport Channel */
    initialize ( ... );
    while ( for_ever ) {
        message = receive_message ( ... );
        switch ( message.operation ) {
            case LOAD:/* Load a Stream Group and fill the Stream Pipes */
                state = LOADED;
                /* set number of independent streams in the Stream_Group*/
                nSreams = message.hStream_Group.nStreams;
                /* Create a separate Stream Transport Channel for data flow */
                hStream_Channel = Create_Stream_Channel (message.hStream_Group);
                /*  Reply to the Remote Stream Controller and return the Stream Channel*/
                *message.phStream_Channel = hStream_Channel;
                reply_message ( message.sender,  ... );
                ...
                break;
            case UNLOAD:  /* Unload a Stream Group and clear the Stream Pipes */
                state = UNLOADED;
                ...
                break;
            case PLAY: /* Start retrieving data and feeding the Stream Pipes */
                state = PLAYING;
                ...
                break;
            case PAUSE:    /* Stop retrieving data and feeding the Stream Pipes */
                state = LOADED;
                ...
                break;
            ...
            ...
```

```
        }
        if ( state == PLAYING ) {
            int  i;
            ...
            for ( i = 0; i <= nStreams; i++ ) {
                ...
                Transmit ( hStream_Channel, next presentation unit );
                ...
            }
            ...
        }
    }
}   /* End Remote_Network_Stream_IO_Manager ( ) */
```

Appendix K

Local Network Stream I/O Manager

```
Local_Network_Stream_IO_Manager ( ... ) {
int  nStreams;   /* Number of independent Streams in a Stream Group */
IOMGR_MSG  message;    /* Stream I/O Manager Message structure */
int  hStream_Channel;   /* Handle to the Stream Transport Channel */
    initialize ( ... );
    while ( for_ever ) {
        message = receive_message ( ... );
        switch ( message ) {
            case LOAD:/* Load a Stream Group and fill the Stream Pipes */
                state = LOADED;
                /* set number of independent streams in the Stream_Group*/
                nSreams = message.hStream_Group.nStreams;
                /* Receive a separate Stream Transport Channel for data flow */
                hStream_Channel = message.hStream_Channel;
                /*  Find and connect to the Remote Network Stream I/O Manager */
                hRemote_Network_Stream_IO_Manager = find (hStream_Channel, ... );
                connect (hRemote_Network_Stream_IO_Manager, ... );
                ...
                break;
            case UNLOAD:  /* Unload a Stream Group and clear the Stream Pipes */
                state = UNLOADED;
                ...
                break;
            case PLAY: /* Start retrieving data and feeding the Stream Pipes */
                state = PLAYING;
                ...
                break;
            case PAUSE:    /* Stop retrieving data and feeding the Stream Pipes */
                state = LOADED;
                ...
                break;
            ...
            ...
```

```
        }
        If ( state == PLAYING ) {
            ...
            for ( i = 0; i <= nStreams; i++ ) {
                ...
                Enqueue ( next presentation unit );
                ...
            }
            ...
            Feed_Back ( );
            ...
        }
    }
}   /* End Local_Network_Stream_IO_Manager ( ) */
```

Appendix L

Implicit Timing Rate Control

```
int     p = 0;   /* consumed presentation units */
int     t = 0;   /* reference time base */
int     T;       /* Nominal presentation duration */
int     D;       /* Requested presentation duration */
Transmit ( ... ) {
boolean done = FALSE;
while (!done) {
    if ((p*T <= t) && (t < (p+1)*T)) {
        /* Consume and transmit the next presentation unit */
        Consume_and_Transmit ( next presentation unit);
        p = p + 1;
        done = TRUE;
    }
    if ((p+1)*T <= t) {
        /* Adjust the video rate by transmitting null presentation units */
        Fabricate ( null presentation unit );
        Consume_and_Transmit ( null presentation unit );
        p = p + 1;
    }
}
/* Increment virtual stream time */
t = t + D;
}   /* End Transmit ( ) */
```

Appendix M

Explicit Timing Rate Control

```
int     p;      /* presentation time */
int     d;      /* presentation duration */
int     t = 0;  /* reference virtual time base */
int     D;      /* Requested presentation duration */
Transmit ( ... ) {
boolean done = FALSE;
while (!done) {
    p = presentation_time ( next presentation unit );
    d = presentation_duration ( next presentation unit );
    if ((p <= t) && (t < (p + d))) {
        /* Consume and transmit the next presentation unit */
        Consume_and_Transmit ( next presentation unit );
        done = TRUE;
    }
    if ((p + d) <= t) {
        /* Adjust the video rate by transmitting null presentation units */
        Fabricate ( null presentation unit );
        Consume_and_Transmit ( null presentation unit );
    }
}
/* Increment virtual stream time */
t = t + D;
}   /* End Transmit ( ) */
```

M1

Appendix N

Adaptive Load Balancing with Feedback

```
#define Ncycles          < number of cycles over which average is calculated >
#define Nbands           < number of bands the pipe size is divided into >
int cycles = 0;          /* count of cycles for averaging */
int average_sum = 0;     /* count of running sum for average calculation */
int Stream_Pipe_size;    /* size of Stream Pipe measured in presentation units */
int previous_average_pipe_size_index = Nbands;
int Rate_Table[Nbands];  /* Table for converting pipe size index to desired rate */
Feed_Back ( ) {
    boolean feedback = FALSE;
    int average_pipe_size;
    int average_pipe_size_index;
    if ( cycles ) {
        average_sum = average_sum + Stream_Pipe_size;
        cycles--;
    }
    else {
        average_pipe_size = (average_sum / Ncycles) * 100;
        average_pipe_size_index = average_pipe_size / Nbands;
        if (average_pipe_size_index < (previous_average_pipe_size_index - 1)) {
            feedback = TRUE;
            presentation_data_rate = Rate_Table[average_pipe_size_index];
        }
        if (average_pipe_size_index > (previous_average_pipe_size_index + 1)) {
            feedback = TRUE;
            presentation_data_rate = Rate_Table[average_pipe_size_index];
        }
        previous_average_pipe_size = average_pipe_size;
        cycles = Ncycles;
    }
    if ( feedback ) {
        callback ( hLocal_Stream_Controller, FEEDBACK, presentation_data_rate );
    }
```

What is claimed is:

1. A computer-based media data processor for controlling the timing of computer processing of digitized continuous time-based media data composed of a sequence of presentation units, each unit characterized by a prespecified presentation duration during a computer presentation of the media data, the media processor comprising:

a reference clock which indicates a start time of presentation processing of the media data presentation units and which maintains a current presentation time as the media data presentation unit sequence is processed for presentation;

a counter for counting each presentation unit in the presentation unit sequence after that presentation unit is processed for presentation, to maintain a current presentation unit count; and

a comparator connected to the reference clock and the counter, and programmed with the prespecified presentation duration, the comparator comparing a product of the presentation unit duration and the current presentation unit count, specified by the counter, with the current presentation time, specified by the reference clock, after each presentation unit is processed for presentation, and based on the comparison, releasing a next sequential presentation unit to be processed for presentation when the product matches the current presentation time count, and deleting a next sequential presentation descriptor in that sequence when the product exceeds the current presentation time count.

2. The media data processor of claim 1 wherein the media data presentation unit sequence comprises a video frame sequence including a plurality of intracoded video frames.

3. The media data processor of claim 2 wherein each frame of the video frame sequence comprises an intracoded video frame.

4. The media data processor of claim 3 wherein the video frame sequence comprises a motion JPEG video sequence.

5. The media data processor of claim 2 wherein each of the plurality of intracoded video frames comprises a key frame and is followed by a plurality of corresponding non-key frames, each key frame including media data information required for presentation of the following corresponding non-key frames.

6. The media data processor of claim 1 further comprising a flow controller, connected to said comparator, for receiving an indication from the comparator that a presentation unit should be released for presentation, determining availability of a next presentation unit in the presentation unit sequence to be processed, and based on that availability, generating and releasing a virtual presentation unit of the prespecified presentation duration to be presented as a default presentation unit in place of a next presentation unit when a next presentation unit is not available and until the next presentation unit is available.

7. The media data processor of claim 6 wherein the flow controller is adapted to monitor and identify a previously unavailable presentation unit when that unit is later available, and in response to identification of the later available unit, withholding the unit from release for presentation, whereby the later available unit is not presented.

8. The media data processor of claim 6 wherein the media data presentation unit sequence comprises a motion JPEG video sequence, the presentation units comprise video frames, and wherein each virtual presentation unit comprises a most recently presented video frame.

9. The media data processor of claim 1 wherein the media data presentation unit sequence comprises an audio sequence.

10. The media data processor of claim 1 wherein said clock is adapted to indicate a start time of presentation processing of a plurality of media data presentation unit sequences, the start time being common to the plurality of sequences, and which maintains a current presentation time as the media data sequences are processed for presentation;

a counter for counting each presentation unit in the plurality of presentation unit sequences after that presentation unit is processed for presentation, to maintain a distinct current presentation unit count for each presentation unit sequence; and

a comparator connected to the reference clock and the counter, and programmed with the prespecified presentation duration, the comparator comparing for each of the plurality of presentation unit sequences a product of the presentation unit duration and the current presentation unit count of that sequence, specified by the counter, with the current presentation time, specified by the reference clock, after each presentation unit from that sequence is processed for presentation, and based on the comparison, releasing a next sequential presentation unit in that presentation unit sequence to be processed for presentation when the product matches the current presentation time count, and deleting a next sequential presentation unit in that presentation unit sequence when the product exceeds the current presentation time count, whereby the plurality of media data presentation unit sequences are synchronously processed for simultaneous synchronous presentation.

11. The media data processor of claim 10 wherein the plurality of media data presentation unit sequences comprise an intracoded video frame sequence and an audio sequence.

12. A computer-based media data processor for controlling the computer presentation of digitized continuous time-based media data composed of a sequence of presentation units, each unit characterized by a prespecified presentation duration and presentation time during a computer presentation of the media data and further characterized as a distinct media data type, the media data processor comprising:

a media data input manager for retrieving media data from a corresponding media data access location in response to a request for computer presentation of specified presentation unit sequences, determining the media data type of each presentation unit in the retrieved media data, designating each retrieved presentation unit to a specified media data presentation unit sequence based on the media data type determination for that presentation unit, assembling a sequence of presentation descriptors for each of the specified presentation unit sequences, each presentation descriptor comprising presentation unit media data for one designated presentation unit in that sequence, all presentation descriptors in an assembled sequence being of a common media data type, associating each presentation descriptor with a corresponding presentation duration and presentation time, based on the retrieved media data, and linking the presentation descriptors in each assembled sequence to establish a progression of presentation units in each of the sequences; and

a media data interpreter, connected to the media data input manager, for accepting from the media data input manager the assembled presentation descriptor sequences one descriptor at a time and releasing the sequences for presentation one presentation unit at a time, indicating a start time of presentation processing of the presentation unit sequences, maintaining a current presentation time as the sequences are processed

for presentation, counting each unit in the sequences after that unit is released to be processed for presentation, to maintain a distinct current presentation unit count for each sequence, comparing for each of the presentation unit sequences a product of the presentation unit duration and the current presentation unit count of that sequence with the currently maintained presentation time after each unit from that sequence is processed for presentation, and based on the comparison, releasing for presentation processing a next sequential presentation unit in that sequence when the product matches the currently maintained presentation time count and deleting a next sequential presentation unit in that presentation unit sequence when the product exceeds the currently maintained presentation time count.

13. The media data processor of claim 12 wherein the media data access location comprises a computer storage location.

14. The media data processor of claim 13 further comprising a presentation unit sequence controller for initiating the media data input manager and the media data interpreter, specifying to the media data input manager and the media data interpreter the presentation unit sequences to be presented, and controlling starting and stopping of sequence presentation in response to user specifiedation.

15. The media data processor of claim 13 wherein the specified media data presentation unit sequences comprise a video frame sequence including a plurality of intracoded video frames.

16. The media data processor of claim 15 wherein each frame of the video frame sequence comprises an intracoded video frame.

17. The media data processor of claim 16 wherein the video frame sequence comprises a motion JPEG video sequence.

18. The media data processor of claim 15 wherein each of the plurality of intracoded video frames comprises a key frame and is followed by a plurality of corresponding non-key frames, each key frame including media data information required for presentation of the following corresponding non-key frames.

19. The media data processor of claim 16 wherein the specified media data presentation unit sequences comprise a motion JPEG video sequence and an audio sequence.

20. The media processor of claim 14 wherein the media data interpreter further determines for each specified presentation unit sequence availability of a next presentation descriptor when based on said comparison a next presentation unit should be released for presentation, and based on that availability, generates and releases a virtual presentation unit of the prespecified presentation duration to be presented as a default presentation unit each time a next presentation unit in that sequence is not available for presentation and until the next presentation unit is available.

21. The media processor of claim 20 wherein the local media data interpreter is adapted to monitor and identify a previously unavailable presentation unit when that descriptor is later available, and in response to identification of the later available descriptor, withholding the later available presentation unit from release for presentation, whereby the later available presentation unit is not presented.

22. The media data processor of claim 20 wherein the plurality of media data presentation unit sequences comprises an intracoded video sequence of video frames and an audio sequence of audio samples, and wherein each virtual video presentation unit comprises a most recently presented video frame and each virtual audio presentation unit comprises a silent audio sample.

23. The media data processor of claim 12 wherein the media data retrieved by the media data input manager comprises a storage presentation unit sequence composed of presentation units for the specified presentation unit sequences, presentation units of the specified presentation unit sequences being alternately interleaved to compose the storage presentation unit sequence.

24. The media data processor of claim 12 wherein the media data retrieved by the media data input manager comprises a plurality of storage presentation unit sequences, each storage presentation unit sequence composed of presentation units for a specified presentation unit sequence and all presentation units in a storage presentation unit sequence being of a common media data type.

25. The media data processor of claim 24 wherein the start time of presentation processing indicated by the media data interpreter is common to all of the specified presentation unit sequences, whereby the specified presentation unit sequences are synchronously processed for simultaneous synchronous presentation.

26. The media data processor of claim 25 wherein the specified presentation unit sequences comprise a video presentation unit sequence of intracoded video frames and an audio presentation unit sequence of audio samples, and wherein the media data interpreter prioritizes audio presentation units over video presentation units by generating and releasing a virtual video frame to be presented as a default presentation unit each time a next presentation unit is not available for presentation and until the next presentation unit is available, the virtual video frame comprising a most recently presented video frame.

27. The media data processor of claim 14 wherein the retrieved media data presentation units are encoded in a storage code and compressed, and further comprising a presentation system for decoding the presentation units, decompressing the presentation units, and converting the digitized presentation units to a corresponding analog representation for presentation.

28. The media data processor of claim 12 wherein the media data interpreter maintains the current presentation time at a prespecified time rate such that presentation units of the specified presentation sequences are each presented for a presentation duration different than the prespecified presentation duration.

29. The media data processor of claim 12 wherein the media data interpreter monitors for each specified presentation unit sequence an actual presentation rate of the presentation descriptors in that sequence released for presentation, compares the actual presentation rate with a prespecified nominal presentation rate, and based on the comparison, generates and releases a virtual presentation unit of the prespecified presentation duration to be presented as a default presentation unit each time the monitored presentation rate is greater than the prespecified presentation rate, and based on the comparison, skips over a presentation unit each time the monitored presentation rate is less than the prespecified presentation rate.

30. A computer-based method for controlling the timing of computer processing of digitized continuous time-based media data composed of a sequence of presentation units, each unit characterized by a prespecified presentation duration during a computer presentation of the media data, the method comprising:

indicating a start time of presentation processing of the media data presentation units;

maintaining a current presentation time as the media data presentation unit sequence is processed for presentation;

counting each presentation unit in the presentation unit sequence after that presentation unit is processed for presentation, to maintain a current presentation unit count; and

comparing a product of the presentation unit duration and the current presentation unit count with the current presentation time after a presentation unit is processed for presentation, and based on the comparison, releasing a presentation unit next in the presentation unit sequence to be processed for presentation when the product matches the current presentation time count, and deleting a presentation unit next in the presentation unit sequence when the product exceeds the current presentation time count.

31. The media data processor of claim **30** wherein the specified media data presentation unit sequence comprise a video frame sequence including a plurality of intracoded video frames.

32. The media data processor of claim **31** wherein each frame of the video frame sequence comprises an intracoded video frame.

33. The media data processor of claim **32** wherein the video frame sequence comprises a motion JPEG video sequence.

34. The media data processor of claim **31** wherein each of the plurality of intracoded video frames comprises a key frame and is followed by a plurality of corresponding non-key frames, each key frame including media data information required for presentation of the following corresponding non-key frames.

35. The method of claim **30** further comprising:

determining the availability of a next presentation unit in the presentation unit sequence to be processed, and based on that availability, generating and releasing a virtual presentation unit of the prespecified presentation duration to be presented as a default presentation unit in place of the next presentation unit when a next presentation unit is not available and until the next presentation unit is available.

36. The method of claim **35** further comprising:

identifying a previously unavailable presentation unit when that unit is later available; and

in response to the identification of the later available unit, withholding the unit from release for presentation, whereby the later available unit is not presented.

37. A computer-based method for controlling the computer presentation of digitized continuous time-based media data composed of a sequence of presentation units, each unit characterized by a prespecified presentation duration and presentation time during a computer presentation of the media data and further characterized as a distinct media data type, the method comprising:

retrieving media data from a computer storage location in response to a request for computer presentation of specified presentation unit sequences;

determining the media data type of each presentation unit in the retrieved media data;

designating each retrieved presentation unit to a specified media data presentation unit sequence based on the media data type determination for that presentation unit;

assembling a sequence of presentation descriptors for each of the specified presentation unit sequences, each descriptor comprising media data for one designated presentation unit in that sequence, each sequence of presentation descriptors being of a common media data type;

associating each presentation descriptor with a corresponding presentation duration and presentation time, based on the retrieved media data;

linking the presentation descriptors of each sequence to establish a progression of presentation units in that sequence;

indicating a start time of presentation processing of the presentation descriptor sequences;

maintaining a current presentation time as the sequences are processed for presentation;

counting each presentation unit in the media data sequences after that unit is processed for presentation, to maintain a distinct current presentation unit count for each sequence;

comparing for each of the presentation unit sequences a product of the presentation unit duration and the current presentation unit count of that sequence with the current presentation time after each presentation unit from that sequence is processed for presentation, and based on the comparison, releasing a presentation unit next in that presentation unit sequence to be processed for presentation when the product matches the current presentation time count, and deleting a presentation unit next in that presentation unit sequence when the product exceeds the current presentation time count.

38. The method of claim **37** wherein the retrieved media data comprises a storage presentation unit sequence composed of presentation units for the specified presentation unit sequences, presentation units of the specified presentation unit sequences being alternately interleaved to compose the storage presentation unit sequence.

39. The method of claim **38** wherein the start time of presentation processing is common to all of the specified presentation unit sequences, whereby the specified presentation unit sequences are synchronously processed for simultaneous synchronous presentation.

40. The media data processor of claim **39** wherein the specified media data presentation unit sequences comprise a video frame sequence including a plurality of intracoded video frames.

41. The media data processor of claim **40** wherein each frame of the video frame sequence comprises an intracoded video frame.

42. The media data processor of claim **1** wherein each of the plurality of intracoded video frames comprises a key frame and is followed by a plurality of corresponding non-key frames, each key frame including media data information required for presentation of the following corresponding non-key frames.

43. The media data processor of claim **41** wherein the specified media data presentation unit sequences comprise a motion JPEG video sequence and an audio sequence.

44. A computer-based media data processor for controlling transmission of digitized media data in a packet switching network, the media data comprising a sequence of continuous time-based presentation units, each unit characterized by a prespecified presentation duration and presentation time during a computer presentation of the media data and further characterized as a distinct media data type, the network comprising a plurality of client computer processing nodes interconnected via packet-based data distribution channels, the media data processor comprising:

a remote media data controller for receiving from a client processing node a request for presentation of specified presentation unit sequences;

a remote media data input manager for receiving from the remote media data controller an indication of the specified presentation unit sequences, and in response to the request, retrieving media data from a corresponding media access location, determining the media data type of each presentation unit in the retrieved media data, designating each retrieved presentation unit to a specified media data presentation unit sequence based on the media data type determination for that presentation unit, assembling a sequence of presentation descriptors for each of the specified presentation unit sequences, each descriptor comprising media data for one designated presentation unit in that sequence, all presentation descriptors in an assembled sequence being of a common media data type, associating each presentation descriptor with a corresponding presentation duration and presentation time, based on the retrieved media data, and linking the descriptors in each assembled sequence to establish a profession of presentation units in each of the specified presentation unit sequences;

a remote network media data manager connected to the remote media data input manager, for accepting from the remote media data manager the assembled specified presentation descriptor sequences one presentation descriptor at a time, assembling transmission presentation unit packets each composed of at least a portion of a presentation descriptor and its media data, all presentation descriptors and media data in an assembled packet being of a common media data type, and releasing the assembled packets for transmission via the network to the client processing node requesting presentation of the specified presentation unit sequences;

a local media data controller for transmitting the request for presentation of specified presentation unit sequences from the client processing node to the remote media data controller via the network and controlling starting and stopping of sequence presentation in response to user specifiedations;

a local network media data manager for receiving from the local media data controller an indication of the specified presentation unit sequences, receiving the transmission presentation unit packets transmitted from the remote network media data manager via the network, designating a presentation unit sequence for each presentation descriptor and its media in the received packets to thereby assemble the presentation descriptor sequences each corresponding to one specified presentation unit sequence, all presentation descriptors and media data in an assembled sequence being of a common media data type, and linking the descriptors in each assembled sequence to establish a progression of presentation units for each of the presentation unit sequences; and

a local media data interpreter, connected to the local network media data manager, for accepting the assembled presentation descriptor sequences one descriptor at a time and releasing the sequences for presentation one presentation unit at a time, indicating a start time of presentation processing of the sequences, maintaining a current presentation time as the sequences are processed for presentation, and based on the presentation duration of each presentation unit, synchronizing presentation of the specified presentation unit sequences with the current presentation time.

45. The media data processor of claim **44** wherein the specified media data presentation unit sequences comprise a video frame sequence including a plurality of intracoded video frames.

46. The media data processor of claim **45** wherein each frame of the video frame sequence comprises an intracoded video frame.

47. The media data processor of claim **46** wherein the video frame sequence comprises a motion JPEG video sequence.

48. The media data processor of claim **45** wherein each of the plurality of intracoded video frames comprises a key frame and is followed by a plurality of corresponding non-key frames, each key frame including media data information required for presentation of the following corresponding non-key frames.

49. The media data processor of claim **45** wherein the specified presentation unit sequences comprise a motion JPEG video sequence and an audio sequence.

50. The media data processor of claim **44** wherein the media access location comprises a computer storage location.

51. The media data processor of claim **50** wherein the computer storage location comprises a computer file.

52. The media data processor of claim **44** wherein the local media data interpreter synchronizes presentation of the specified presentation unit sequences by comparing for each of the presentation descriptors in each of the presentation descriptor sequences the presentation time corresponding to that descriptor with the currently maintained presentation time, and based on the comparison, releasing a next sequential presentation unit to be processed for presentation when the corresponding presentation time of that descriptor matches the current presentation time, and deleting a next sequential presentation unit to be processed for presentation when the current presentation time exceeds the corresponding presentation time of that descriptor.

53. The media data processor of claim **44** wherein the local media data interpreter synchronizes presentation of the specified presentation unit sequences by counting each presentation unit in the sequences after that presentation unit is released to be processed for presentation, to maintain a distinct current presentation unit count for each sequence, comparing for each of the presentation unit sequences a product of the presentation unit duration and the current presentation unit count of that sequence with the currently maintained presentation time after a presentation unit from that sequence is released to be processed for presentation, and based on the comparison, releasing a next sequential presentation unit in that presentation unit sequence when the product matches the currently maintained presentation time, and deleting a next sequential presentation unit in that presentation unit sequence when the product exceeds the currently maintained presentation time.

54. The media data processor of claim **52** wherein the local media data interpreter determines for each presentation descriptor sequence availability of a next sequential presentation descriptor in that sequence when the currently maintained presentation time indicates that a presentation unit should be released for presentation, and based on that availability, generates and releases a virtual presentation unit of the corresponding presentation duration to be presented as a default presentation unit each time a next presentation descriptor in that sequence is not available and until a next presentation descriptor is available.

55. The media data processor of claim 53 wherein the local media data interpreter determines for each presentation descriptor sequence availability of a next sequential presentation descriptor in that sequence when based on said comparison a presentation unit should be released for presentation, and based on that availability, generates and releases a virtual presentation unit of the corresponding presentation duration to be presented as a default presentation unit each time a next presentation descriptor in that sequence is not available and until a next presentation descriptor is available.

56. The media data processor of either of claims 54 or 55 wherein the local media data interpreter is adapted to monitor and identify a previously unavailable presentation descriptor when that descriptor is later available, and in response to identification of the later available descriptor, withholding the later available presentation unit from release for presentation, whereby the later available unit is not presented.

57. The media data processor of either of claims 54 or 55 wherein the specified presentation unit sequences comprises a motion video sequence of video frames and an audio sequence of audio samples, and wherein each virtual video presentation unit comprises a most recently presented video frame and each virtual audio presentation unit comprises silent audio samples.

58. The media data processor of either of claims 54 or 55 wherein the specified presentation unit sequences comprise an audio sequence and a video frame sequence composed of a plurality of key video frames, each key frame followed by a plurality of corresponding non-key frames, each key frame including media data information required for presentation of the following corresponding non-key frames, and wherein the local media data interpreter is adapted to monitor and identify a previously unavailable presentation descriptor corresponding to a key frame when that descriptor is later available, and in response to identification of the later available key frame descriptor, withholding the descriptor and any following descriptors, corresponding to non-key frames following the key frame, from release for presentation, whereby the later available key frame and following non-key frames are not presented.

59. The media data processor of claim 50 wherein the media data retrieved by the remote media data input manager comprises a plurality of storage presentation unit sequences, each storage presentation unit sequence composed of presentation units for a specified presentation unit sequence and all presentation units in a storage presentation unit sequence being of a common media data type, and wherein the start time of presentation processing indicated by the local media data interpreter is common to all of the specified presentation descriptor sequences, whereby the presentation unit sequences are synchronously processed for simultaneous synchronous presentation.

60. The media data processor of claim 50 wherein the network comprises a local area network.

61. The media data processor of claim 50 wherein the network comprises a wide area network.

62. The media data processor of claim 60 wherein the remote media data controller advertises to client computer processing nodes, via the network, an indication of specified presentation unit sequences that may be requested from that remote media data controller.

63. The media data processor of claim 44 wherein the media access location comprises a digitized representation of analog media data captured in real time.

64. The media data processor of claim 44 wherein the media access location comprises a PBX server.

65. The media data processor of claim 44 wherein presentation of the specified presentation unit sequences comprises display of the presentation unit sequences.

66. The media data processor of claim 44 wherein presentation of the specified presentation unit sequences comprises VCR tape printing of the presentation unit sequences.

67. The media data processor of claim 65 wherein display of the presentation unit sequences comprises display on a computer monitor.

68. The media data processor of claim 65 wherein display of the presentation unit sequences comprises display on a television monitor.

69. The media data processor of claim 44 wherein presentation of the specified presentation unit sequences comprises recording the sequences at a computer storage location.

70. The media data processor of claim 44 wherein presentation of the specified presentation unit sequences comprises sending the sequences to a PBX server.

71. The media data processor of claim 44 wherein the media access location comprises an access point to a public switch network.

72. The media data processor of claim 44 wherein presentation of the specified presentation unit sequences comprises sending the sequences to an access point in a public switch network.

73. The media data processor of claim 44 wherein the remote media data controller further receives from the local media data controller via the network an indication of a specified presentation data rate at which the specified presentation unit sequences are to be transmitted via the network to the client node, and in response, the remote media data controller indicates the specified presentation data rate to the remote media data input manager and the remote media data network manager;

further wherein the media data retrieved by the remote media data input manager comprises a plurality of storage presentation unit sequences stored in a computer storage location, each storage presentation unit sequence composed of presentation units corresponding to a specified presentation unit sequence and all presentation units in a storage presentation unit sequence being of a common media data type; and

further wherein the remote media data input manager designates each of a portion of the presentation unit descriptors as the descriptor sequences are assembled, the portion including a number of descriptors based on the specified presentation data rate, each designated descriptor comprising null media data, to thereby compose the presentation descriptor sequences with only a portion of storage presentation unit media data, whereby the specified presentation unit sequences attain the specified presentation data rate of transmission.

74. The media data processor of claim 44 wherein the remote media data controller further receives from the local media data controller via the network an indication of a specified presentation data rate at which the specified presentation unit sequences are to be transmitted via the network to the client node, and in response, the remote media data controller indicates the specified presentation data rate to the remote media data input manager and the remote media data network manager;

further wherein the media data retrieved by the remote media data input manager comprises a storage presentation unit sequence stored in a computer storage location, presentation units of the storage presentation

unit sequence comprising alternately interleaved presentation units from the specified presentation unit sequences; and

further wherein the remote network media data manager designates each of a portion of the presentation descriptors as the transmission presentation unit packets are assembled, the portion including a number of descriptors based on the specified presentation data rate, each designated descriptor comprising null media data, to thereby compose the transmission presentation unit packets with only a portion of specified sequence presentation unit media data, whereby the transmission presentation unit packets attain the specified presentation data rate of transmission.

75. The media data processor of either of claims **73** or **74** wherein the specified presentation unit sequences comprise a motion video frame sequence including a plurality of intracoded video frames and an audio sequence.

76. The media data processor of claim **73** wherein the specified presentation unit sequences include an audio sequence composed of audio presentation units having corresponding audio storage presentation units; and

wherein the portion of presentation units having a presentation unit sequence designation includes all audio storage presentation units.

77. The media data processor of claim **74** wherein the specified presentation unit sequences include an audio sequence composed of audio presentation units; and

wherein the portion of presentation units having a transmission presentation unit packet designation includes all audio presentation units.

78. The media data processor of either of claims **73** or **74** wherein the local media data controller receives from the client node a client user-specified indication of a specified presentation data rate at which the specified presentation unit sequences are to be transmitted to the client node.

79. The media data processor of either of claims **73** or **74** wherein the local network media data manager monitors availability of presentation descriptors as they are accepted by the local media data interpreter one descriptor at a time from the local network media data manager, and based on the availability, indicates the specified presentation data rate to the local media data controller for indication to the remote media data controller.

80. The media data processor of claim **79** wherein the local network media data manager indicates a specified presentation data rate that is higher than a current presentation unit sequence transmission rate when the monitored availability increases to prespecified upper availability.

81. The media data processor of claim **79** wherein the local network media data manager indicates a specified presentation data rate that is lower than a current presentation unit sequence transmission rate when the monitored availability decreases to a prespecified lower availability.

82. A method for controlling transmission of digitized media data in a packet switching network, the media data comprising a sequence of continuous time-based presentation units, each unit characterized by a prespecified presentation duration and presentation time during a computer presentation of the media data and further characterized as a distinct media data type, the network comprising a plurality of client computer processing nodes interconnected via packet-based data distribution channels, the method comprising:

receiving from a client processing node a request for presentation of specified presentation unit sequences;

in response to the request, retrieving media data from a corresponding media access location;

determining the media data type of each presentation unit in the retrieved media data;

designating each retrieved presentation unit to a specified media data presentation unit sequence based on the media data type determination for that presentation unit;

assembling a sequence of presentation descriptors for each of the specified presentation unit sequences, each descriptor comprising media data for one designated presentation unit in that sequence, all presentation descriptors in an assembled sequence being of a common media data type;

associating each presentation descriptor with a corresponding presentation duration and presentation time, based on the retrieved media data;

linking the descriptors in each assembled sequence to establish a progression of presentation units in each of the specified presentation unit sequences;

assembling transmission presentation unit packets each composed of at least a portion of a presentation descriptor and its media data, all presentation descriptors and media data in an assembled packet being of a common media data type; and

releasing the assembled packets for transmission via the network to the client processing node requesting presentation of the specified presentation unit sequences.

83. The method of claim **82** further comprising:

receiving at the client processing node the transmission presentation unit packets via the network;

designating a presentation unit sequence for each presentation descriptor and its media data in the received packets to thereby assemble the presentation descriptor sequences each corresponding to one specified presentation unit sequence, all presentation descriptors in an assembled sequence being of a common media data type;

linking the descriptors in each assembled sequence to establish a progression of presentation units for each of the presentation unit sequences;

indicating a start time of presentation processing of the sequences;

maintaining a current presentation time as the descriptor sequences are processed for presentation; and

based on the presentation duration of each presentation unit, synchronizing presentation of the specified presentation unit sequences with the current presentation time.

84. The method of claim **82** wherein the specified presentation unit sequences comprise an intracoded video frame sequence and an audio sequence.

85. The method of claim **83** wherein the step of synchronizing presentation of the specified presentation unit sequences comprises:

comparing for each of the presentation descriptors in each of the presentation descriptor sequences the presentation time corresponding to that descriptor with the currently maintained presentation time; and

based on the comparison, releasing a next sequential presentation unit to be processed for presentation when the corresponding presentation time of that descriptor matches the current presentation time, and deleting a next sequential presentation unit to be processed for presentation when the current presentation time exceeds the corresponding presentation time of that descriptor.

86. The method of claim 83 wherein the step of synchronizing presentation of the specified presentation unit sequences comprises:

counting each presentation descriptor in the sequences after that presentation unit is released to be processed for presentation, to maintain a distinct current presentation unit count for each sequence;

comparing for each of the presentation unit sequences a product of the presentation unit duration and the current presentation descriptor count of that sequence with the currently maintained presentation time after a presentation unit from that sequence is released to be processed for presentation; and

based on the comparison, releasing a next sequential presentation unit in that presentation unit sequence when the product matches the currently maintained presentation time, and deleting a next sequential presentation unit in that presentation unit sequence when the product exceeds the currently maintained presentation time.

87. The method of claim 83 further comprising:

receiving via the network an indication of a specified presentation data rate at which the specified presentation unit sequences are to be transmitted via the network to the client node, further wherein the media data retrieved comprises a plurality of storage presentation unit sequences stored in a computer storage location, each storage presentation unit sequence composed of presentation units corresponding to a specified presentation unit sequence and all presentation units in a storage presentation unit sequence being of a common media data type; and

designating each of a portion of the presentation unit descriptors as the descriptor sequences are assembled, the portion including a number of descriptors based on the specified presentation data rate, each designated descriptor comprising null media data, to thereby compose the presentation descriptor sequences with only a portion of storage presentation unit media data, whereby the specified presentation unit sequences attain the specified presentation data rate of transmission.

88. The method of claim 83 further comprising:

receiving via the network an indication of a specified presentation data rate at which the specified presentation unit sequences are to be transmitted via the network to the client node, further wherein the media data retrieved comprises a storage presentation unit sequence stored in a computer storage location, presentation units of the storage presentation unit sequence comprising alternately interleaved presentation units from the specified presentation unit sequences; and

designating each of a portion of the presentation descriptors as the presentation descriptor sequences are assembled, the portion including a number of descriptors based on the specified presentation data rate, each designated descriptor comprising null media data, to thereby compose the transmission presentation unit packets with only a portion of specified sequence presentation unit media data, whereby the transmission presentation unit packets attain the specified presentation data rate of transmission.

89. The method of either of claims 87 or 88 further comprising:

monitoring availability of presentation descriptors after the descriptors are received at the client node and before the descriptors are presented; and

based on the availability, indicating the specified presentation data rate via the network.

90. A computer-based media data processor for capturing and controlling transmission of digitized media data in a packet switching network, the media data comprising a sequence of continuous time-based presentation units, each unit characterized by a prespecified presentation duration and presentation time during a computer presentation of the media data and further characterized as a distinct media data type, the network comprising a plurality of client computer processing nodes interconnected via packet-based data distribution channels, the media data processor comprising:

a local media data controller for indicating user-specified presentation unit sequences to be captured from a client node for recording at a network media access location;

a local media data interpreter for receiving the specified presentation unit sequences from the client node, assembling a sequence of presentation descriptors for each of the received specified presentation unit sequences, each descriptor comprising media data for one presentation unit in that sequence, all presentation descriptors in an assembled sequence being of a common media data type, associating each presentation descriptor with a corresponding presentation duration and presentation time, based on the retrieved media data, and linking the descriptors in each assembled sequence to establish a progression of presentation units for each of the presentation unit sequences;

a local network media data manager connected to the local media data interpreter, for accepting from the local media data interpreter the assembled specified presentation descriptor sequences one presentation descriptor at a time, assembling transmission presentation unit packets each composed of at least a portion of a presentation descriptor and its media data, all presentation descriptors and media data in an assembled packet being of a common media data type, and releasing the assembled packets for transmission via the network to the network media access location;

a remote media data controller for receiving from the local media data controller an indication of the specified presentation unit sequences to be recorded at the network media access location;

a remote network media data manager for receiving from the remote media data controller an indication of the specified presentation unit sequences, receiving the transmission presentation unit packets transmitted from the local network media data manager via the network, designating a presentation unit sequence for each presentation descriptor and its media data in the received packets to thereby assemble the presentation descriptor sequences each corresponding to one specified presentation unit sequence, all presentation descriptors and media data in an assembled sequence being of a common media data type, and linking the descriptors in each sequence to establish a progression of presentation units for each of the presentation unit sequences; and

a remote media data output manager for receiving from the remote media data controller an indication of the specified presentation unit sequences, and connected to the remote network media data manager, for accepting the assembled presentation descriptor sequences one descriptor at a time, formatting the accepted sequences and media data in a media access format, and releasing the formatted sequences to the media access location.

91. The media processor of claim 90 wherein the media access location comprises a computer storage location.

92. The media processor of claim 91 wherein the computer storage location comprises computer file.

93. The media processor of claim 90 wherein the specified presentation unit sequences comprise an intracoded video frame sequence and an audio sequence.

94. The media processor of claim 93 wherein the media access location comprises a computer file.

95. The media processor of claim 94 wherein the media access format comprises a storage presentation unit sequence, presentation units of the storage presentation unit sequence comprising alternately interleaved presentation units from the specified presentation unit sequences.

96. The media processor of claim 94 wherein the media access format comprises a plurality of storage presentation unit sequences, each storage presentation unit sequence composed of presentation units for a specified presentation unit sequence and all presentation units in a storage presentation unit sequence being of a common media data type.

97. The media processor of claim 93 wherein the media access location comprises a VCR tape printer.

98. A computer-based method for capturing and controlling transmission of digitized media data in a packet switching network, the media data comprising a sequence of continuous time-based presentation units, each unit characterized by a prespecified presentation duration and presentation time during a computer presentation of the media data and further characterized as a distinct media data type, the network comprising a plurality of client computer processing nodes interconnected via packet-based data distribution channels, the method comprising:

indicating user-specified presentation unit sequences to be captured from a client node for recording at a network media access location;

receiving the specified presentation unit sequences from the client node;

assembling a sequence of presentation descriptors for each of the received specified presentation unit sequences, each descriptor comprising media data for one presentation unit in that sequence, all presentation descriptors in an assembled sequence being of a common media data type;

associating each presentation descriptor with a corresponding presentation duration and presentation time, based on the retrieved media data;

linking the descriptors in each assembled sequence to establish a progression of presentation units for each of the presentation unit sequences;

assembling transmission presentation unit packets each composed of at least a portion of a presentation descriptor and its media data, all presentation descriptors and media data in an assembled packet being of a common media data type; and

releasing the assembled packets for transmission via the network to the network media access location.

99. The method of claim 98 further comprising:

receiving the transmission presentation unit packets transmitted via the network;

designating a presentation unit sequence for each presentation descriptor and media data in the received packets to thereby assemble the presentation descriptor sequences each corresponding to one specified presentation unit sequence, all presentation descriptors in an assembled sequence being of a common media data type;

linking the descriptors in each sequence to establish a progression of presentation units for each of the presentation unit sequences;

formatting the accepted sequences and media data in a media access format; and

releasing the formatted sequences to the media access location.

100. The method o claim 99 wherein the media access location comprises a computer storage location.

101. The method of claim 100 wherein the computer storage location comprises computer file.

102. The method of claim 100 wherein the specified presentation unit sequences comprise an intracoded video frame sequence and an audio sequence.

103. A computer-based media data processor for controlling the computer presentation of digitized continuous time-based media data composed of a sequence of presentation units, each unit characterized by a prespecified presentation duration and presentation time during a computer presentation of the media data and further characterized as a distinct media data type, the media data processor comprising:

a media data input manager for retrieving media data from a corresponding media data access location in response to a request for computer presentation of specified presentation unit sequences, determining the media data type of each presentation unit in the retrieved media data, designating each retrieved presentation unit to a specified media data presentation unit sequence based on the media data type determination for that presentation unit, assembling a sequence of presentation descriptors for each of the specified presentation unit sequences, each presentation descriptor comprising media data for one designated presentation unit in that sequence, all presentation descriptors in an assembled sequence being of a common media data type, and linking the presentation descriptors in each assembled sequence to establish a progression of presentation units in each of the sequences; and

a media data interpreter, connected to the media data input manager, for accepting from the media data input manager the assembled presentation descriptor sequences one descriptor at a time and releasing the sequences for presentation one presentation unit at a time, indicating a start time of presentation processing of the presentation unit sequences, maintaining a current presentation time as the sequences are processed for presentation, counting each unit in the sequences after that unit is released to be processed for presentation, to maintain a distinct current presentation unit count for each sequence, comparing for each of the presentation unit sequences a product of the presentation unit duration and the current presentation unit count of that sequence with the currently maintained presentation time after each unit from that sequence is processed for presentation, and based on the comparison, releasing for presentation processing a next sequential presentation unit in that sequence when the product matches the currently maintained presentation time count, and deleting a next sequential presentation unit in that sequence when the product exceeds the currently maintained presentation time count.

104. A computer-based media data processor for controlling transmission of digitized media data in a packet switching network, the media data comprising a sequence of continuous time-based presentation units, each unit characterized by a prespecified presentation duration and presentation time during a computer presentation of the media data and further characterized as a distinct media data type, the network comprising a plurality of client computer process-

ing nodes interconnected via packet-based data distribution channels, the media data processor comprising:

a remote media data controller for receiving from a client processing node a request for presentation of specified presentation unit sequences;

a remote media data input manager for receiving from the media data controller an indication of the specified presentation unit sequences, and in response to the request, retrieving media data from a corresponding media access location, determining the media data type of each presentation unit in the retrieved media data, designating each retrieved presentation unit to a specified media data presentation unit sequence based on the media data type determination for that presentation unit, assembling a sequence of presentation descriptors for each of the specified presentation unit sequences, each descriptor comprising media data for one designated presentation unit in that sequence, all presentation descriptors in an assembled sequence being of a common media data type, and linking the descriptors in each assembled sequence to establish a progression of presentation units in each of the specified presentation unit sequences;

a remote network media data manager connected to the remote media data input manager, for accepting from the remote media data manager the assembled specified presentation descriptor sequences one presentation descriptor at a time, assembling transmission presentation unit packets each composed of at least a portion of a presentation descriptor and its media data, all presentation descriptors and media data in an assembled packet being of a common media data type, and releasing the assembled packets for transmission via the network to the client processing node requesting presentation of the specified presentation unit sequences;

a local media data controller for transmitting the request for presentation of specified presentation unit sequences from the client processing node to the remote media data controller via the network and controlling starting and stopping of sequence presentation in response to user specifiedations;

a local network media data manager for receiving from the local media data controller an indication of the specified presentation unit sequences, receiving the transmission presentation unit packets transmitted from the remote network media data manager via the network, designating a presentation unit sequence for each presentation descriptor and media data in the received packets to thereby assemble the presentation descriptor sequences each corresponding to one specified presentation unit sequence, all presentation descriptors and media data in an assembled sequence being of a common media data type, and linking the descriptors in each assembled sequence to establish a progression of presentation units for each of the presentation unit sequences; and

a local media data interpreter, connected to the local network media data manager, for accepting the assembled presentation descriptor sequences one descriptor at a time and releasing the sequences for presentation one unit at a time, indicating a start time of presentation processing of the sequences, maintaining a current presentation time as the descriptor sequences are processed for presentation, and based on the presentation duration of each presentation unit, synchronizing presentation of the specified presentation unit sequences with the current presentation time.

105. A computer-based media data processor for capturing and controlling transmission of digitized media data in a packet switching network, the media data comprising a sequence of continuous time-based presentation units, each unit characterized by a prespecified presentation duration and presentation time during a computer presentation of the media data and further characterized as a distinct media data type, the network comprising a plurality of client computer processing nodes interconnected via packet-based data distribution channels, the media data processor comprising:

a local media data controller for indicating user-specified presentation unit sequences to be captured from a client node for recording at a network media access location;

a local media data interpreter for receiving the specified presentation unit sequences from the client node, assembling a sequence of presentation descriptors for each of the received specified presentation unit sequences, each descriptor comprising media data for one presentation unit in that sequence, all presentation descriptors in an assembled sequence being of a common media data type, and linking the descriptors in each assembled sequence to establish a progression of presentation units for each of the presentation unit sequences;

a local network media data manager connected to the local media data interpreter, for accepting from the local media data interpreter the assembled specified presentation descriptor sequences one presentation descriptor at a time, assembling transmission presentation unit packets each composed of at least a portion of a presentation descriptor and its media data, all presentation descriptors and media data in an assembled packet being of a common media data type, and releasing the assembled packets for transmission via the network to the network media access location;

a remote media data controller for receiving from the local media data controller for receiving from the local media data controller an indication of the specified presentation unit sequences to be recorded at the network media access location;

a remote network media data manager for receiving from the remote media data controller an indication of the specified presentation unit sequences, receiving the transmission presentation unit packets transmitted from the local network media data manager via the network, designating a presentation unit sequence for each presentation descriptor and media data in the received packets to thereby assemble the presentation descriptor sequences each corresponding to one specified presentation unit sequence, all presentation descriptors in an assembled sequence being of a common media data type, and linking the descriptors in each sequence to establish a progression of presentation units for each of the presentation unit sequences; and

a remote media data output manager for receiving from the remote media data controller an indication of the specified presentation unit sequences, and connected to the remote network media data manager, for accepting the assembled presentation descriptor sequences one descriptor at a time, formatting the accepted sequences and media data in a media access format, and releasing the formatted sequences to the media access location.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,719,786
DATED : February 17, 1998
INVENTOR(S) : Nelson et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

IN THE DRAWINGS: Sheet 5, FIG. 6, reference number 36, the phrase "create stream manager" should read --create stream interpreter--; and the figure labeled "STREAM INTERPRETER" should have a reference number --28--. Sheet 6, FIG. 7, the "INTERLEAVED DISK BUFFERS" should be framed in a broken-line box labeled reference number --100--. IN THE SPECIFICATION: Column 1, lines 61 and 64, each occurrence of the word "dock" should read --clock--. Column 2, lines 53-54, the phrase "when the product exceeds the current presentation time count" should read --when the current presentation time count exceeds the product--. Column 4, lines 25-26, the phrase "when the product exceeds the currently maintained presentation time" should read --when the currently maintained presentation time exceeds the product--. Column 10, line 66, the phrase "video frame 108" should read --video frame 110--; and the phrase "audio frame 110" should read --audio frame 108--. Column 12, line 41, the word "sell" should read --self--. Column 14, line 53, the word "stomps" should read --stamps--. Column 15, lines 30-33, the phrase "when the product of processed units and unit duration exceeds the currently maintained time count" should read --when the currently maintained time count exceeds the product of processed units and unit duration--; line 34, the word "then" should read --when--; and line 48, the word "steam" should read --stream--. Column 19, line 54, the number "82" should be --84--. Column 22, line 3, the number "176" should be --174--; and line 40, the word "some" should read --same--. Column 25, line 42, the word "the" should be deleted.

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,719,786
DATED : February 17, 1998
INVENTOR(S) : Nelson et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

IN THE CLAIMS: Claim 1, column 71, lines 27-28; claim 10, column 72, lines 26-27; claim 30, column 75, lines 16-17; and claim 37, column 76, lines 29-30, each occurrence of the phrase "when the product exceeds the current presentation time count" should read --when the currently maintained presentation time count exceeds the product--. Claim 12, column 73, lines 14-16; and claim 103, column 86, lines 57-59, each occurrence of the phrase "when the product exceeds the currently maintained presentation time count" should read --when the current presentation time count exceeds the product--. Claim 14, column 73, line 26, the word "specifiedation" should read --specification--. Claim 44, column 77, line 22, the word "profession" should read --progression--. Claim 53, column 78, lines 55-56; and claim 86, column 83, lines 18-20, each occurrence of the phrase "when the product exceeds the currently maintained presentation time" should read --when the currently maintained presentation time count exceeds the product--. Claim 100, column 86, line 5, the letter "o" should read --of--. Claim 104, column 87, line 42, the word "specifiedations" should read --specification--. Claim 105, column 88, lines 40-41, delete the second occurrence of the phrase "for receiving from the local media data controller".

Signed and Sealed this

First Day of September, 1998

Attest:

Attesting Officer

BRUCE LEHMAN

Commissioner of Patents and Trademarks

(12) **United States Patent**
Brodersen et al.

(10) **Patent No.: US 6,453,459 B1**
(45) **Date of Patent: Sep. 17, 2002**

(54) **MENU AUTHORING SYSTEM AND METHOD FOR AUTOMATICALLY PERFORMING LOW-LEVEL DVD CONFIGURATION FUNCTIONS AND THEREBY EASE AN AUTHOR'S JOB**

(75) Inventors: **Rainer Brodersen**, Santa Clara; **Gregory Kent Wallace**, Palo Alto, both of CA (US)

(73) Assignee: **Apple Computer, Inc.**, Cupertino, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/010,267**

(22) Filed: **Jan. 21, 1998**

(51) **Int. Cl.**[7] **G06F 9/44**

(52) **U.S. Cl.** **717/100**; 707/501.1; 345/719; 345/723; 386/46

(58) **Field of Search** 345/418, 302, 345/474, 326, 719–726; 705/51; 707/522, 501.1; 386/97, 46, 4, 125–126; 709/227; 717/100–116, 136, 149

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,450,489	A *	9/1995	Ostrover et al.	705/51
5,515,490	A *	5/1996	Buchanan et al.	707/500.1
5,544,305	A *	8/1996	Ohmaye et al.	345/776
5,574,843	A *	11/1996	Gerlach, Jr.	345/418
5,592,602	A *	1/1997	Edmunds et al.	345/474
5,619,636	A *	4/1997	Sweat et al.	707/500.1
5,659,793	A *	8/1997	Escobar et al.	707/500.1
5,691,972	A	11/1997	Tsuga et al.	707/500.1
5,694,548	A *	12/1997	Baugher et al.	709/227
5,778,142	A *	7/1998	Taira et al.	386/97
5,892,507	A *	4/1999	Moorby et al.	707/500.1
5,907,704	A *	5/1999	Gudmunson et al.	717/100
6,199,082	B1 *	3/2001	Ferrel et al.	707/522

FOREIGN PATENT DOCUMENTS

EP	0 843 312 A2	5/1998	
EP	0 877 377 A1	11/1998	
GB	2 309 805 A	8/1997	
GB	2 309 805 *	8/1997	..... G06F/9/44
WO	WO 94/28480	12/1994	

OTHER PUBLICATIONS

Uesaka, Y.; "DVD Authoring System," National Technical Report, vol. 42, No. 5, Oct. 1, 1996, pp. 90–96; figures 1–7.
Hughes, K.; "The Tools and Twists of DVD Authoring," Emedia Professional, Online Inc., ISN 1090–946X, vol. 10 No. 12, 12/97, pp. 37–40, 44–48. 50.
Ginige, Athula, et al.; "Hypermedia Authoring," IEEE Multimedia, vol. 2, No. 4, 1221/95, pp. 24–35.

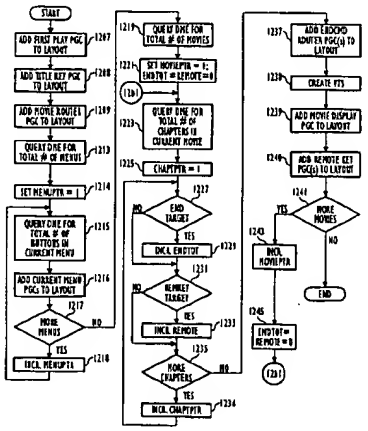
(List continued on next page.)

Primary Examiner—Tuan Q. Dam
Assistant Examiner—Hoang-Vu Antony Nguyen-Ba
(74) *Attorney, Agent, or Firm*—Pillsbury Winthrop LLP

(57) **ABSTRACT**

A DVD authoring system in a processor-based system removes an author from consideration of the DVD Specification during authoring. According to a preferred embodiment, the authoring system provides an authoring engine having an interactive graphical-authoring interface, a data management engine, an emulator, a compiler, a multiplexer and a simulator. Using summary authoring data, the compiler builds a skeleton-form PGC layout structure comprising control PGC abstractions and router PGC abstractions. The compiler then resolves the PGC abstractions according to source-target connections. During playback on a DVD player, the PGC abstractions form elements in a connection-switching abstraction superstructure. Accordingly, in response to DVD-consumer and other control events, a source PGC preferably determines target PGC information and then transfers control, via virtual connections through necessary router PGC abstractions, to a target PGC abstraction. The target PGC abstraction then correspondingly initiates playback of a movie chapter or displays a menu.

4 Claims, 18 Drawing Sheets



OTHER PUBLICATIONS

Sugiyama, Kenji et al.; "Elements of a New Authoring System for Digital Video Disk (DVD)" SMPTE Journal, vol. 106, No. 11; Nov. 1, 1997, pp. 762–767.

Guenette, R.; "Authorware and Iconauthor. Power for Multimedia CD–ROM," CD ROM Professional, Online, Inc., vol. 8, No. 10, Oct. 1, 1995, pp. 81–82, 84, 86–88.

Ryu, S.W., et al.; "A Hierarchical Layered Model for DVD Authoring System," IEEE Transactions on Consumer Electronics, vol. 42, No. 3, Aug. 1, 1996, pp. 814–819.

Sonic Solutions, Instruction Manual, "Premastering in the Age of DVD; A Primer for Creating Content for DVD" 1997.

DVD–Video Software Production Guidebook (Japan Version), DVD Forum, Pioneer Electronic corporation, May 1997.

Daikin U.S. Comtec Laboratories, Reference Manual, "Scenarist 2, The Ultimate Solution for DVD," undated.

* cited by examiner

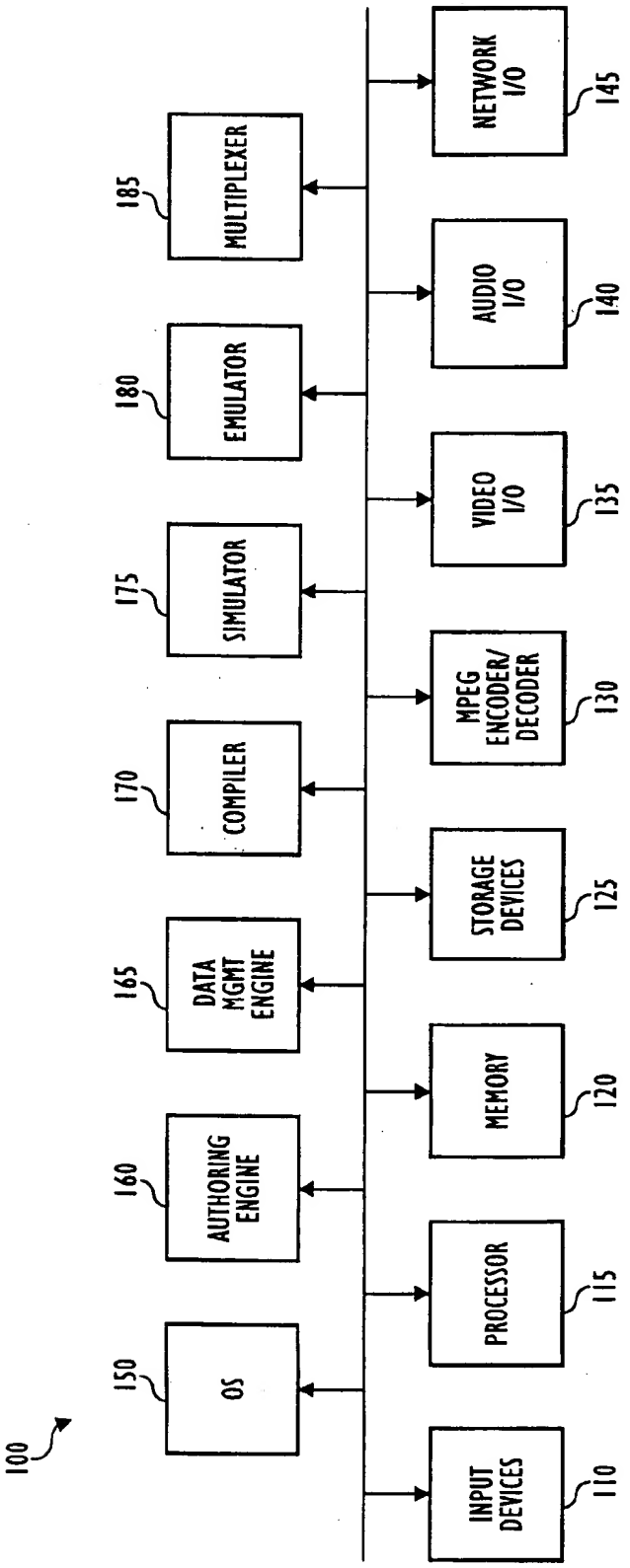

100
OS
150
AUTHORING ENGINE
160
DATA MGMT ENGINE
165
COMPILER
170
SIMULATOR
175
EMULATOR
180
MULTIPLEXER
185
INPUT DEVICES
110
PROCESSOR
115
MEMORY
120
STORAGE DEVICES
125
MPEG ENCODER/ DECODER
130
VIDEO I/O
135
AUDIO I/O
140
NETWORK I/O
145

FIG. 1

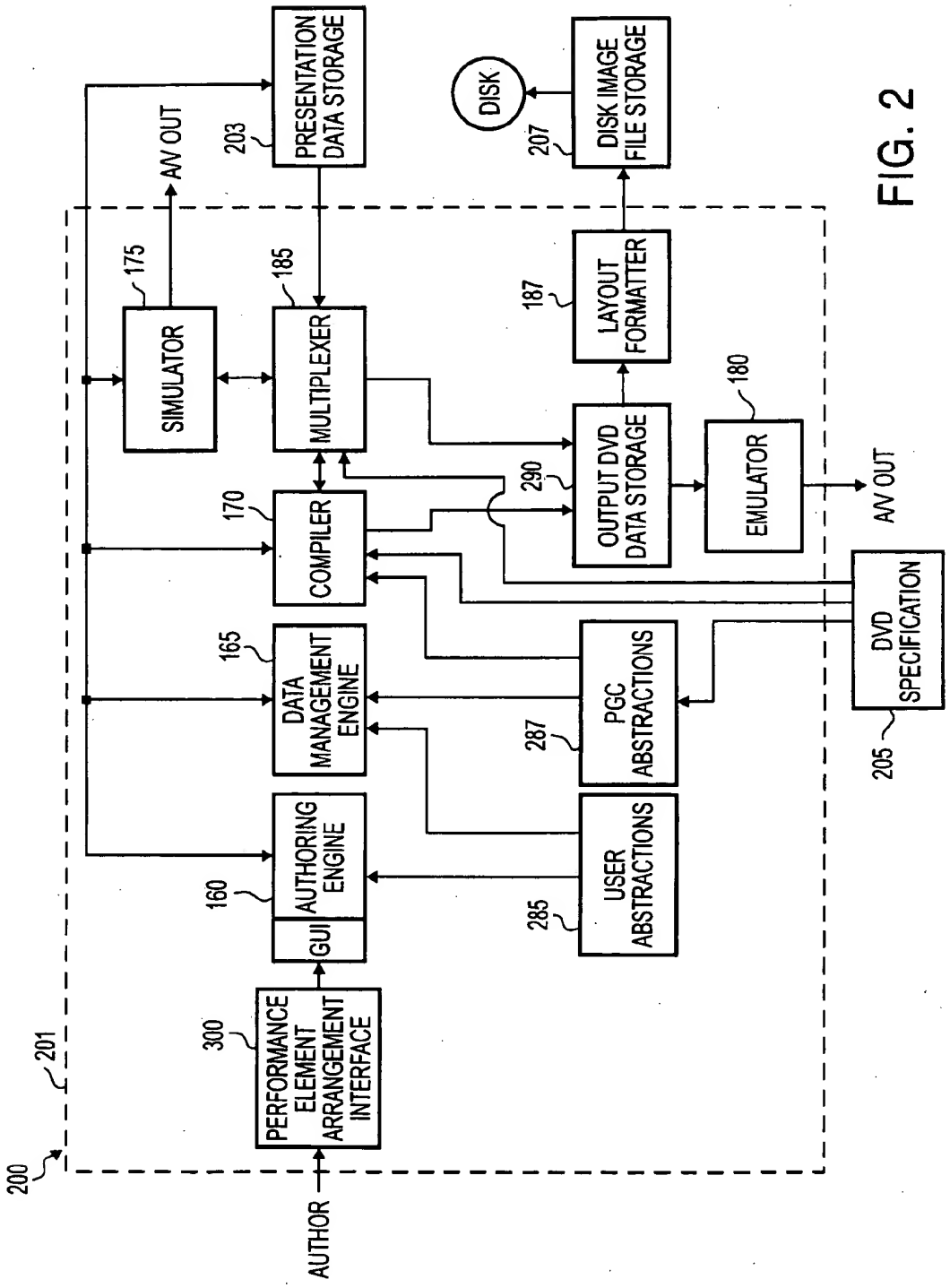

200
201
AUTHOR
PERFORMANCE ELEMENT ARRANGEMENT INTERFACE
300
GUI
AUTHORING ENGINE
160
DATA MANAGEMENT ENGINE
165
COMPILER
170
SIMULATOR
175
MULTIPLEXER
185
A/V OUT
PRESENTATION DATA STORAGE
203
USER ABSTRACTIONS
285
PGC ABSTRACTIONS
287
OUTPUT DVD DATA STORAGE
290
LAYOUT FORMATTER
187
DISK IMAGE FILE STORAGE
207
DISK
EMULATOR
180
A/V OUT
DVD SPECIFICATION
205

FIG. 2

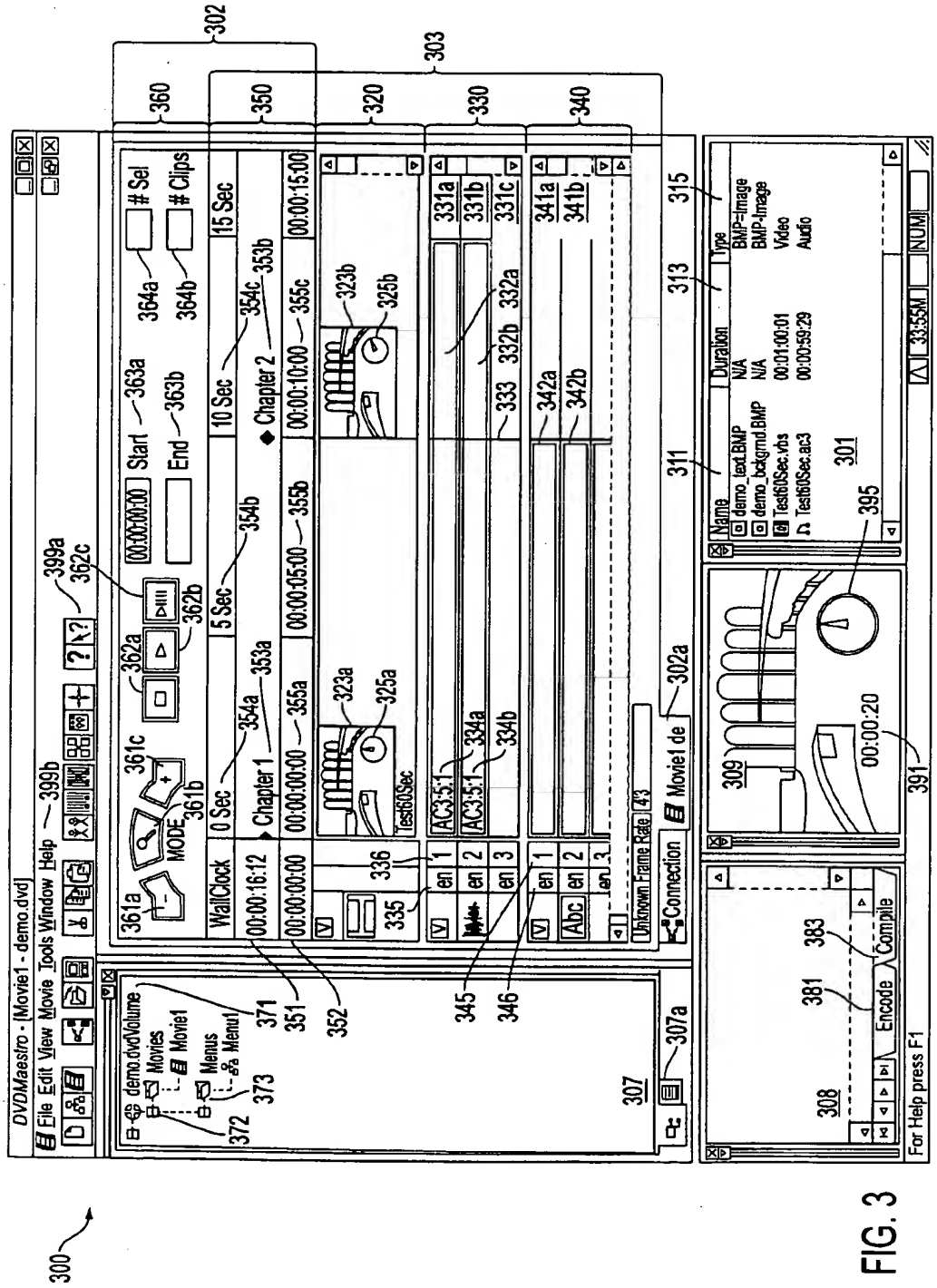

DVDMaestro - [Movie1 - demo.dvd]
File Edit View Movie Tools Window Help
demo.dvdVolume
Movies
Movie1
Menus
Menu1
WallClock
0 Sec
5 Sec
10 Sec
15 Sec
Start
End
Sel
Clips
Chapter 1
Chapter 2
Test60Sec
Movie1 de
Connection
Unknown Frame Rate
Name
Type
Duration
demo_text.BMP
demo_bckgrnd.BMP
Test60Sec.vts
Test60Sec.ac3
BMP-Image
Video
Audio
00:00:20
Encode
Compile
For Help press F1
NUM
300
301
302
303
307
308
309
311
313
315
320
330
340
350
360
371
373
381
383
391
395

FIG. 3

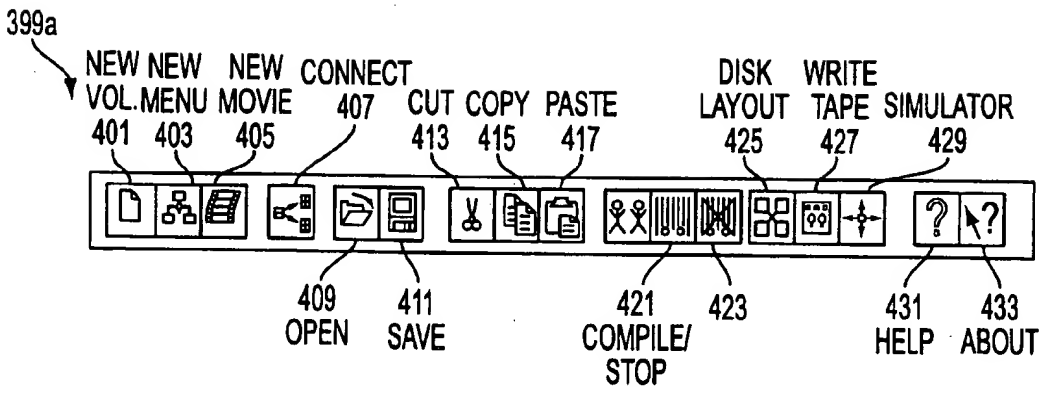

399a
NEW VOL. 401
NEW MENU 403
NEW MOVIE 405
CONNECT 407
CUT 413
COPY 415
PASTE 417
DISK LAYOUT 425
WRITE TAPE 427
SIMULATOR 429
409 OPEN
411 SAVE
421 COMPILE/ STOP
423
431 HELP
433 ABOUT

FIG. 4

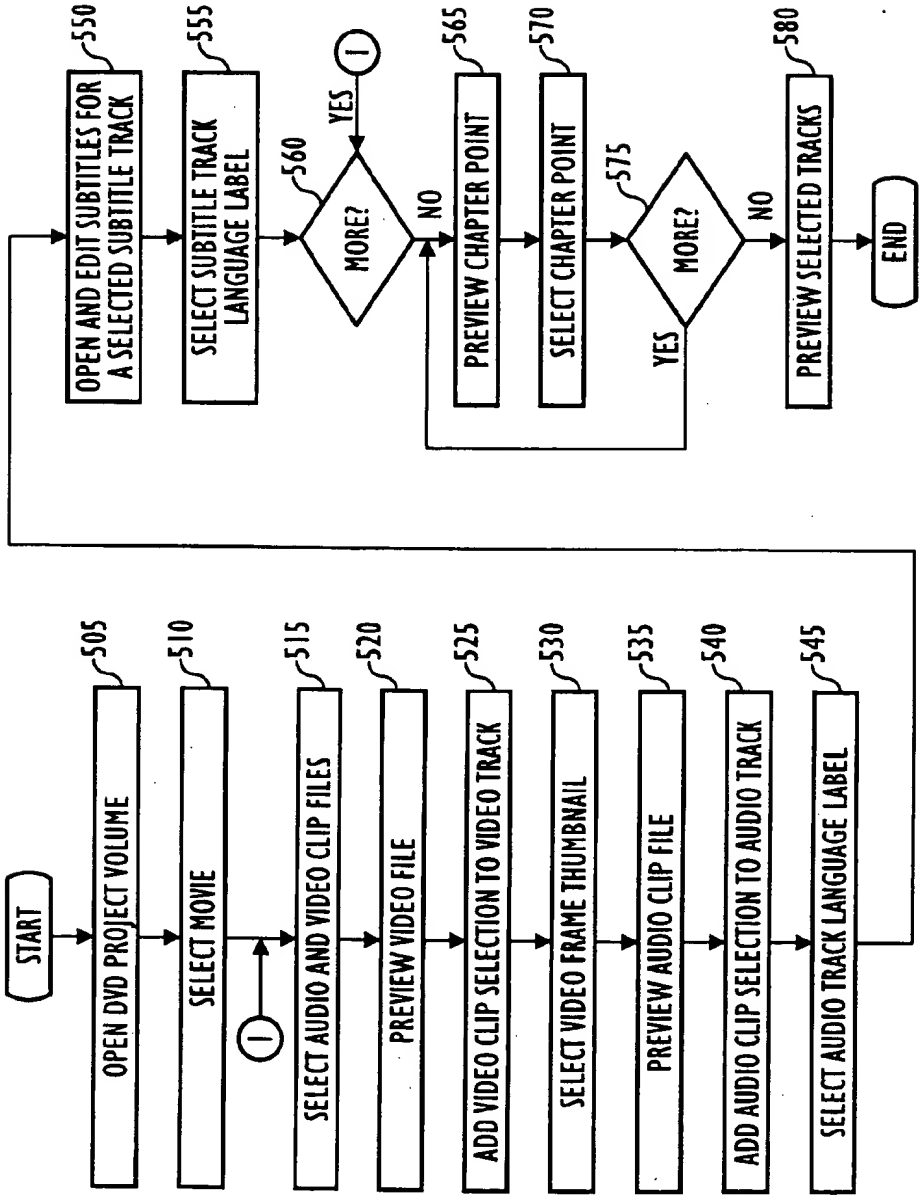

START
OPEN DVD PROJECT VOLUME
505
SELECT MOVIE
510
1
SELECT AUDIO AND VIDEO CLIP FILES
515
PREVIEW VIDEO FILE
520
ADD VIDEO CLIP SELECTION TO VIDEO TRACK
525
SELECT VIDEO FRAME THUMBNAIL
530
PREVIEW AUDIO CLIP FILE
535
ADD AUDIO CLIP SELECTION TO AUDIO TRACK
540
SELECT AUDIO TRACK LANGUAGE LABEL
545
OPEN AND EDIT SUBTITLES FOR A SELECTED SUBTITLE TRACK
550
SELECT SUBTITLE TRACK LANGUAGE LABEL
555
MORE?
560
YES
1
NO
PREVIEW CHAPTER POINT
565
SELECT CHAPTER POINT
570
MORE?
575
YES
NO
PREVIEW SELECTED TRACKS
580
END

FIG. 5

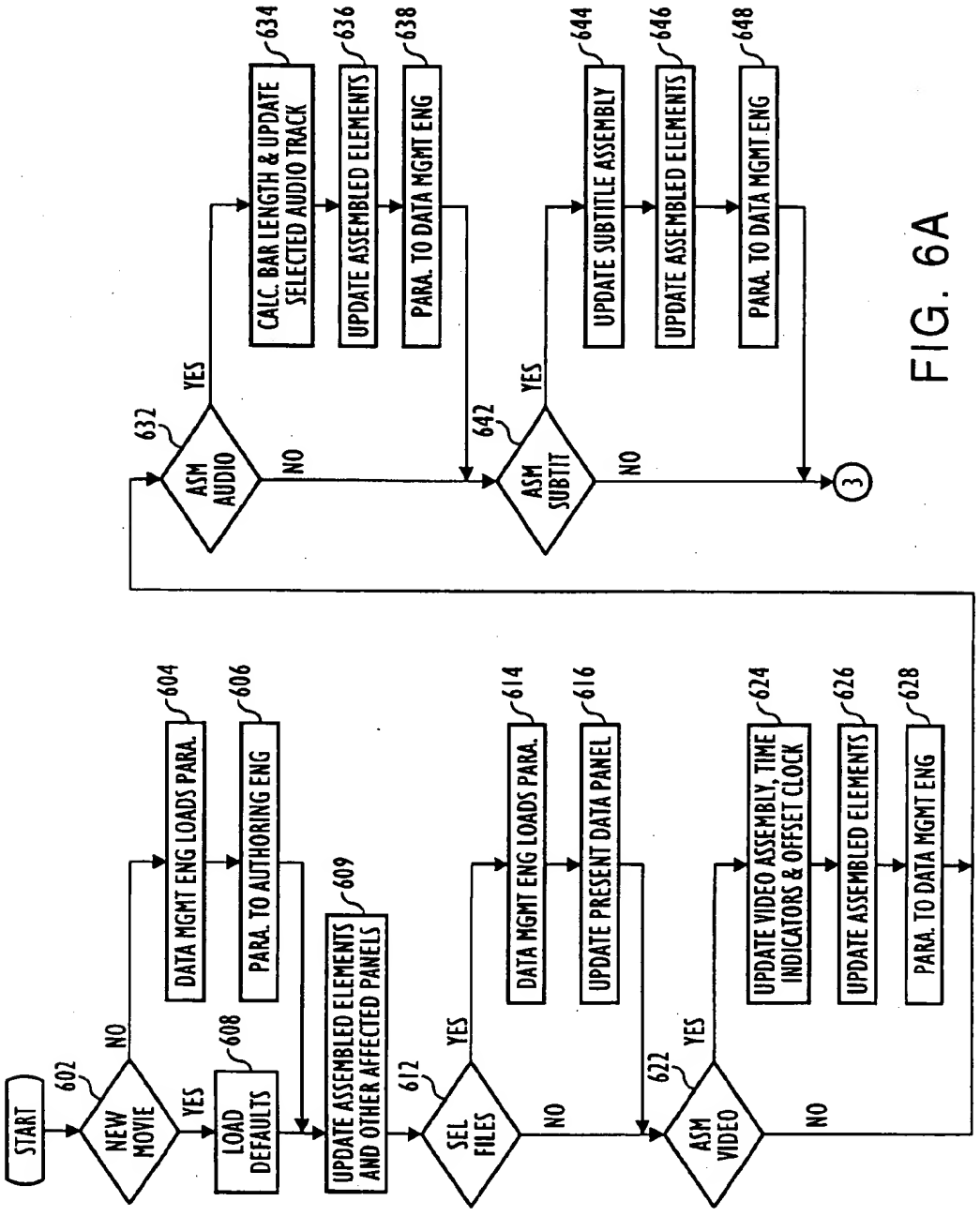

START
602
NEW MOVIE
NO
YES
604
DATA MGMT ENG LOADS PARA.
606
PARA. TO AUTHORING ENG
608
LOAD DEFAULTS
609
UPDATE ASSEMBLED ELEMENTS AND OTHER AFFECTED PANELS
612
SEL FILES
YES
NO
614
DATA MGMT ENG LOADS PARA.
616
UPDATE PRESENT DATA PANEL
622
ASM VIDEO
YES
NO
624
UPDATE VIDEO ASSEMBLY, TIME INDICATORS & OFFSET CLOCK
626
UPDATE ASSEMBLED ELEMENTS
628
PARA. TO DATA MGMT ENG
632
ASM AUDIO
YES
NO
634
CALC. BAR LENGTH & UPDATE SELECTED AUDIO TRACK
636
UPDATE ASSEMBLED ELEMENTS
638
PARA. TO DATA MGMT ENG
642
ASM SUBTIT
YES
NO
644
UPDATE SUBTITLE ASSEMBLY
646
UPDATE ASSEMBLED ELEMENTS
648
PARA. TO DATA MGMT ENG
3

FIG. 6A

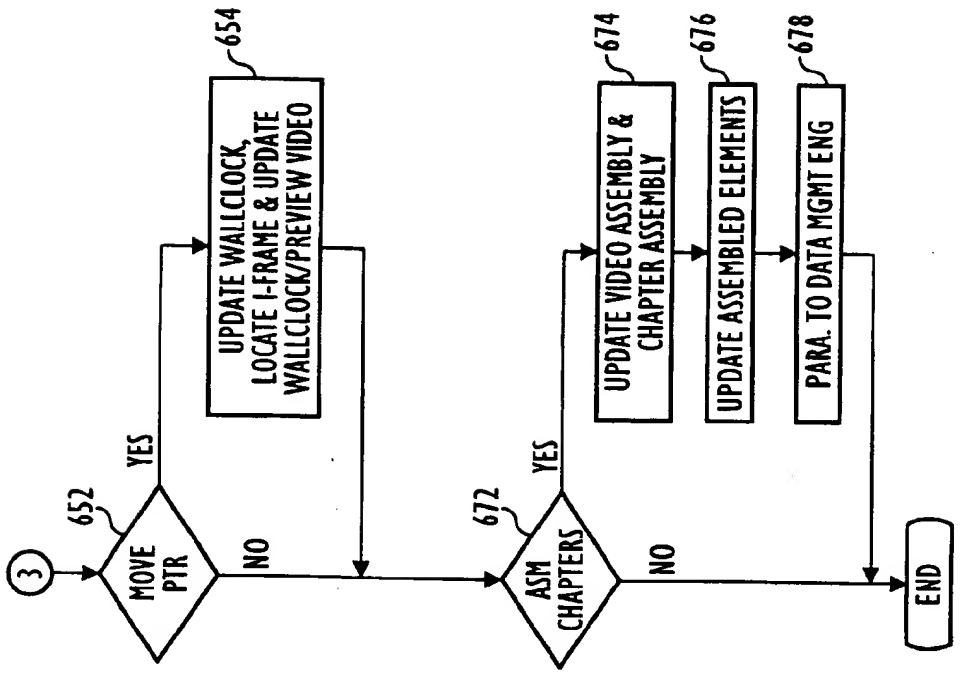

3
652
MOVE PTR
YES
NO
654
UPDATE WALLCLOCK, LOCATE I-FRAME & UPDATE WALLCLOCK/PREVIEW VIDEO
672
ASM CHAPTERS
YES
NO
674
UPDATE VIDEO ASSEMBLY & CHAPTER ASSEMBLY
676
UPDATE ASSEMBLED ELEMENTS
678
PARA. TO DATA MGMT ENG
END

FIG. 6B

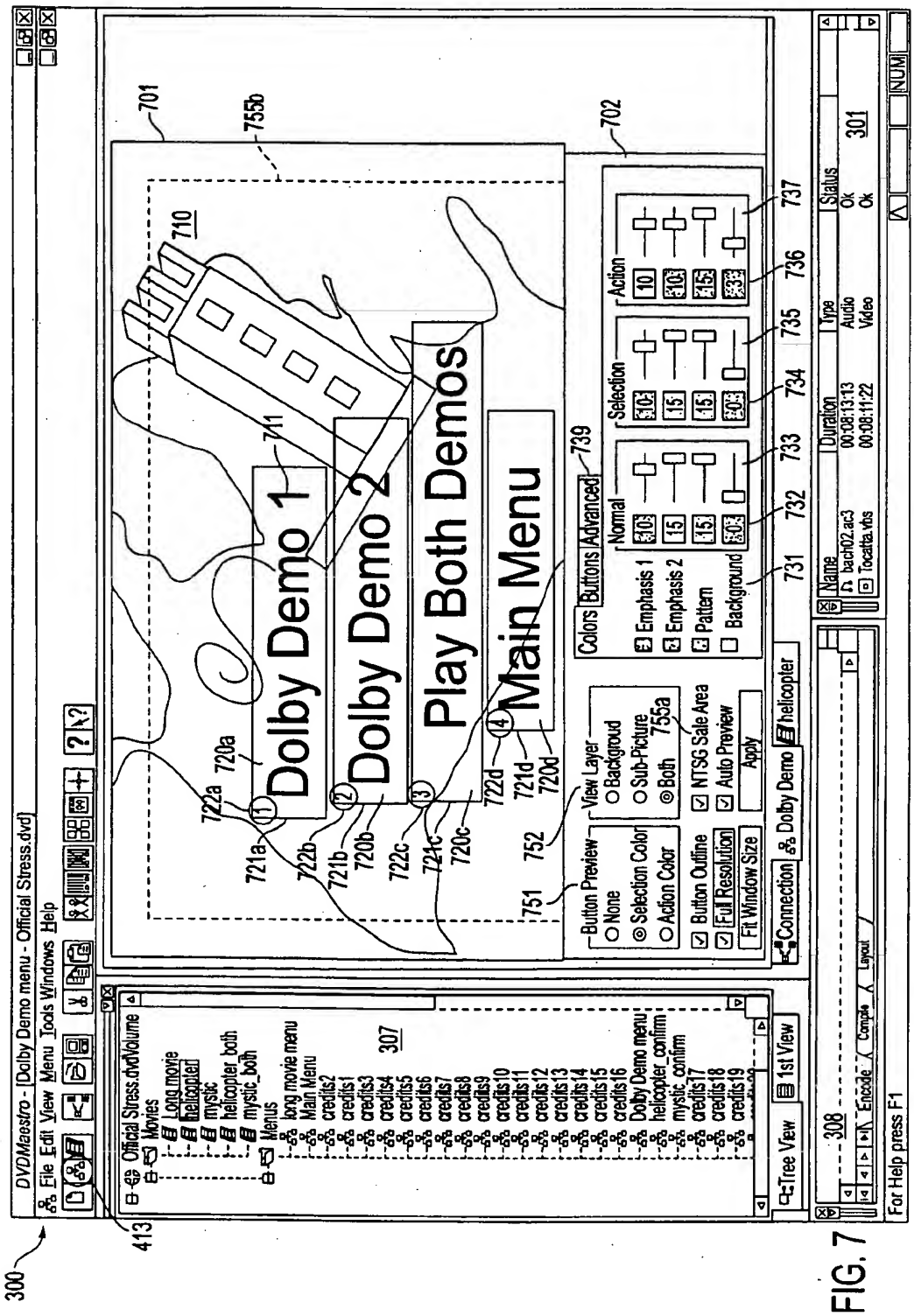

DVDMaestro - [Dolby Demo menu - Official Stress.dvd]
File Edit View Menu Tools Windows Help
Official Stress.dvdVolume
Movies
Long movie
Helicopter
mystic
helicopter_both
mystic_both
Menus
long movie menu
Main Menu
credits1
credits2
credits3
credits4
credits5
credits6
credits7
credits8
credits9
credits10
credits11
credits12
credits13
credits14
credits15
credits16
Dolby Demo menu
helicopter_confirm
mystic_confirm
credits17
credits18
credits19
Tree View
1st View
Dolby Demo 1
Dolby Demo 2
Play Both Demos
Main Menu
Colors
Buttons
Advanced
Normal
Selection
Action
Emphasis 1
Emphasis 2
Pattern
Background
Button Preview
None
Selection Color
Action Color
View Layer
Background
Sub-Picture
Both
NTSG Safe Area
Auto Preview
Button Outline
Full Resolution
Fit Window Size
Apply
Connection
Dolby Demo
helicopter
Name
Duration
Type
Status
bach02.ac3
00:08:13:13
Audio
Ok
Toccatta.vts
00:08:11:22
Video
Ok
For Help press F1
NUM
300
301
307
308
413
701
702
710
711
720a
720b
720c
720d
721a
721b
721c
721d
722a
722b
722c
722d
731
732
733
734
735
736
737
739
751
752
755a
755b

FIG. 7

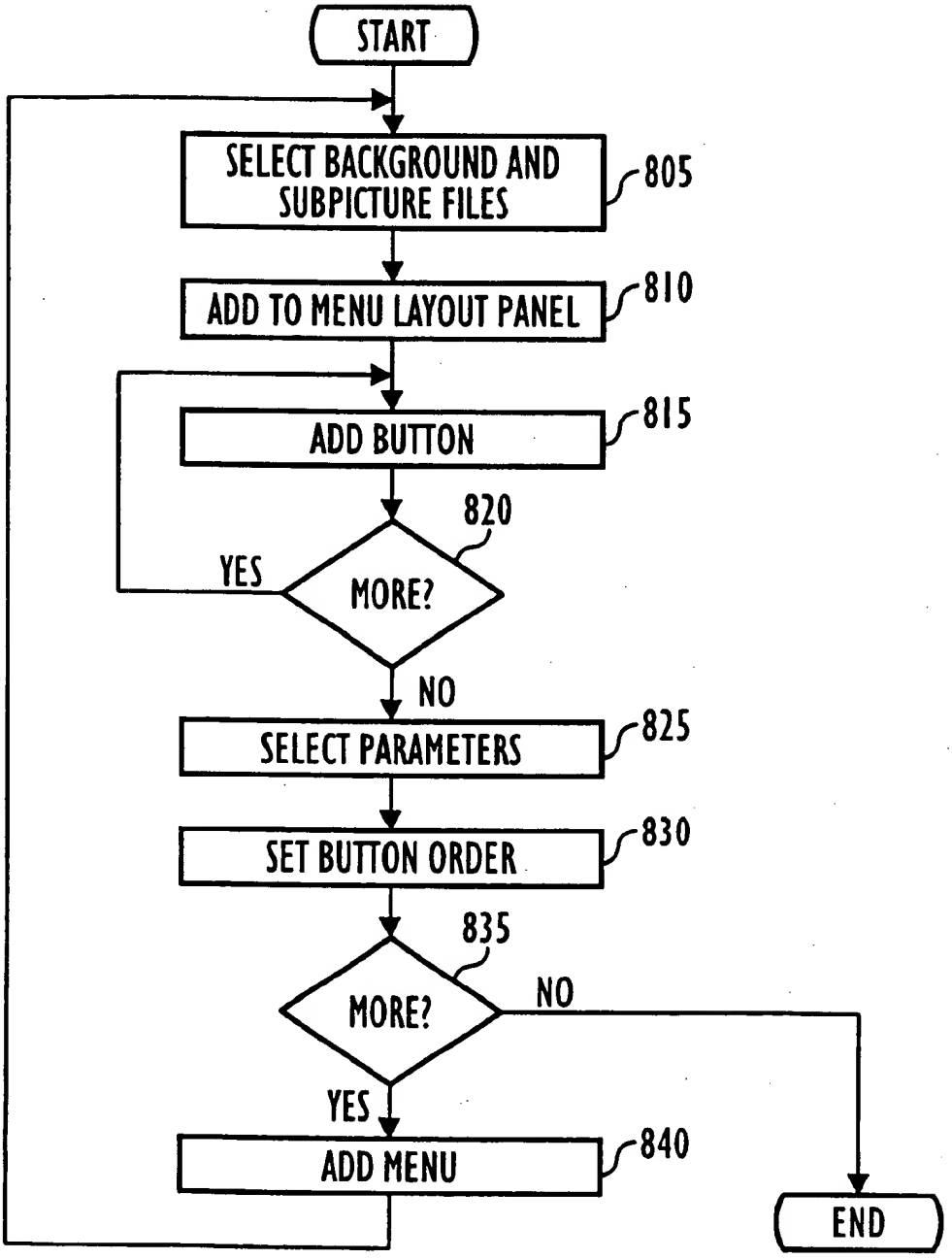

START
SELECT BACKGROUND AND SUBPICTURE FILES
805
ADD TO MENU LAYOUT PANEL
810
ADD BUTTON
815
820
MORE?
YES
NO
SELECT PARAMETERS
825
SET BUTTON ORDER
830
835
MORE?
NO
YES
ADD MENU
840
END

FIG. 8

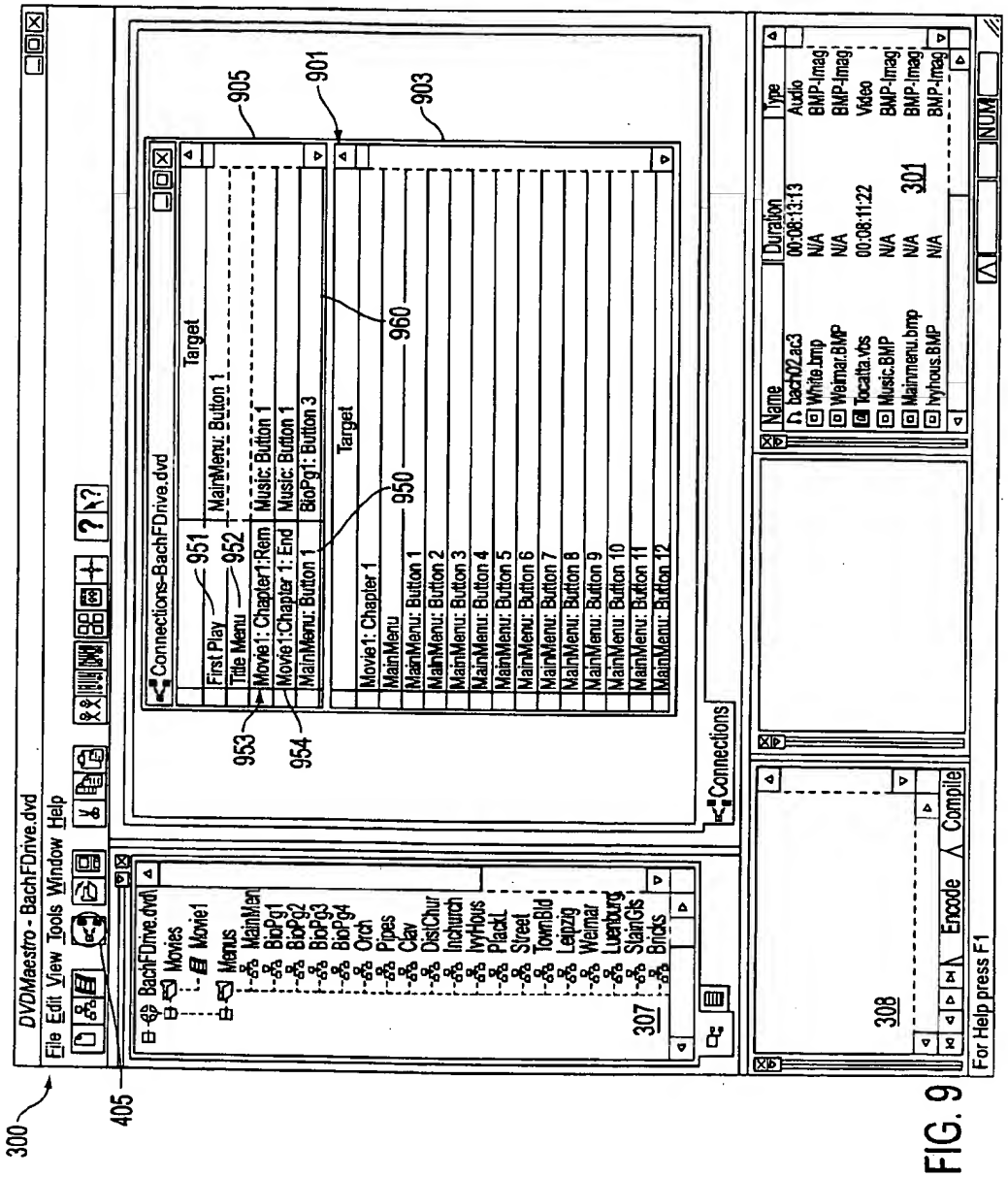

DVDMaestro - BachFDrive.dvd
File Edit View Tools Window Help
BachFDrive.dvd
Movies
Movie1
Menus
MainMen
BioPg1
BioPg2
BioPg3
BioPg4
Orch
Pipes
Clav
DistChur
Inchurch
Ivyhous
PlackL
Street
TownBld
Leipzig
Weimar
Luenburg
StainGls
Bricks
307
Connections-BachFDrive.dvd
Target
First Play
Title Menu
Movie1: Chapter1:Rem
Movie1:Chapter 1: End
MainMenu: Button 1
MainMenu: Button 1
Music: Button 1
Music: Button 1
BioPg1: Button 3
951
952
953
954
905
901
903
960
950
Movie1: Chapter 1
MainMenu
MainMenu: Button 1
MainMenu: Button 2
MainMenu: Button 3
MainMenu: Button 4
MainMenu: Button 5
MainMenu: Button 6
MainMenu: Button 7
MainMenu: Button 8
MainMenu: Button 9
MainMenu: Button 10
MainMenu: Button 11
MainMenu: Button 12
Connections
Name
Duration
Type
bach02.ac3
00:08:13:13
Audio
White.bmp
N/A
BMP-Imag
Weimar.BMP
N/A
BMP-Imag
Toccatta.vbs
00:08:11:22
Video
Music.BMP
N/A
BMP-Imag
Mainmenu.bmp
N/A
BMP-Imag
Ivyhous.BMP
N/A
BMP-Imag
301
NUM
308
Encode
Compile
For Help press F1
300
405

FIG. 9

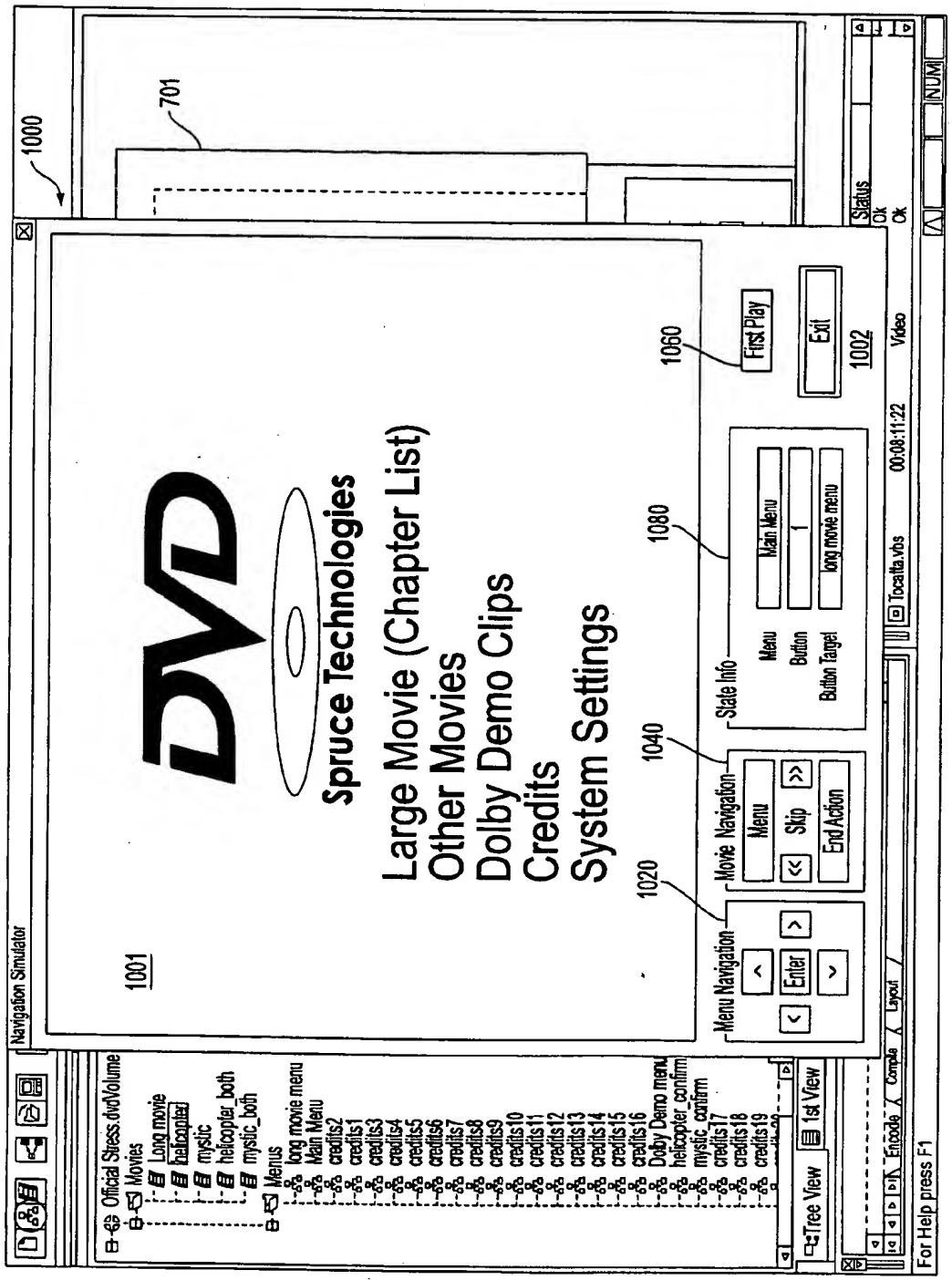

1000
701
1001
DVD
Spruce Technologies
Large Movie (Chapter List)
Other Movies
Dolby Demo Clips
Credits
System Settings
1060
First Play
Exit
1002
1080
State Info
Menu
Button
Button Target
Main Menu
long movie menu
1040
Movie Navigation
Menu
Skip
End Action
1020
Menu Navigation
Enter
Navigation Simulator
Official Stress dvdVolume
Movies
Long movie
helicopter
mystic
helicopter_both
mystic_both
Menus
long movie menu
Main Menu
credits2
credits1
credits3
credits4
credits5
credits6
credits7
credits8
credits9
credits10
credits11
credits12
credits13
credits14
credits15
credits16
Dolby Demo menu
helicopter_confirm
mystic_confirm
credits17
credits18
credits19
Tree View
1st View
Encode
Compile
Layout
Tocatta.vbs
00:08:11:22
Video
Status
Ok
Ok
NUM
For Help press F1

FIG. 10

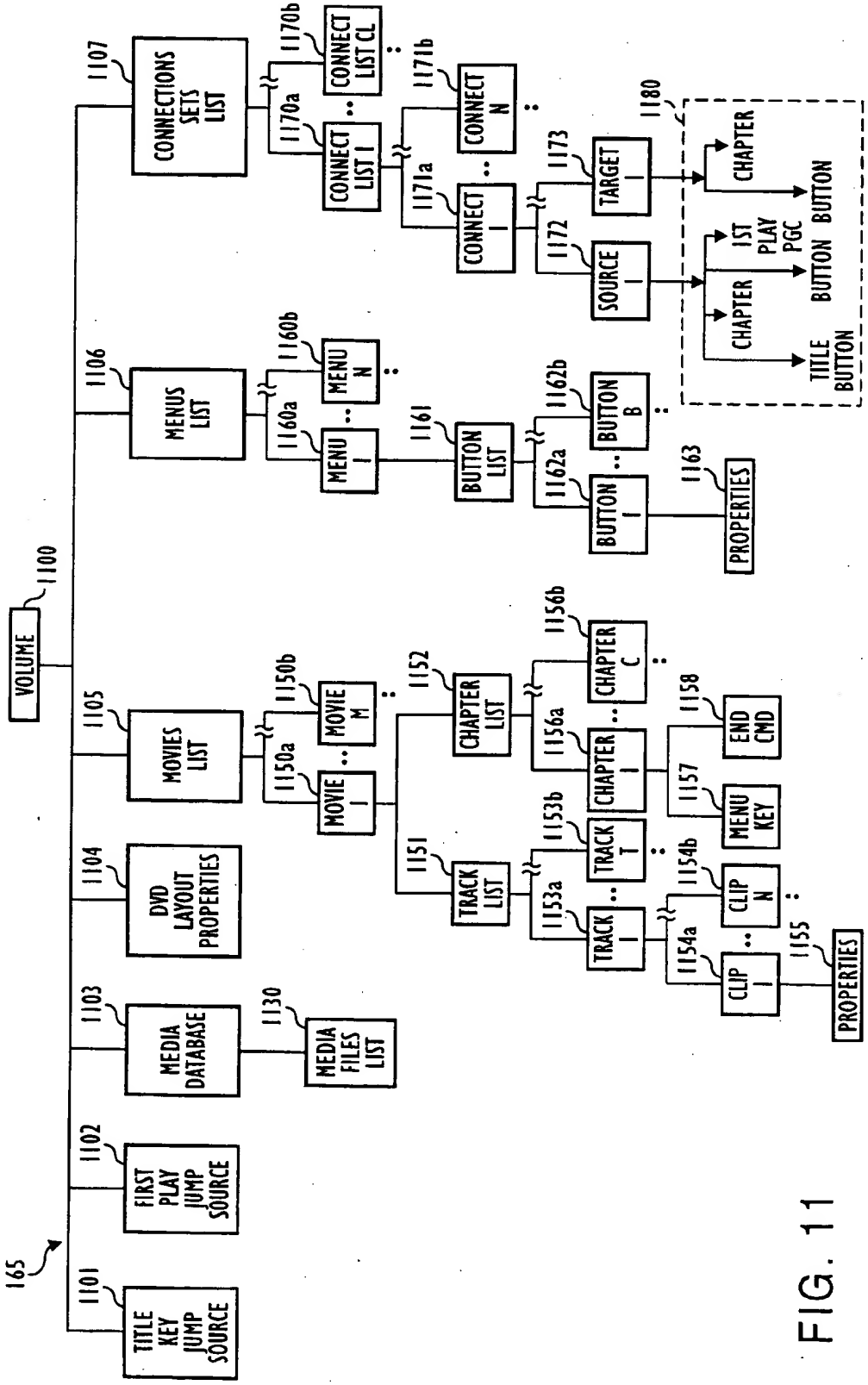

165
VOLUME
1100
TITLE KEY JUMP SOURCE
1101
FIRST PLAY JUMP SOURCE
1102
MEDIA DATABASE
1103
MEDIA FILES LIST
1130
DVD LAYOUT PROPERTIES
1104
MOVIES LIST
1105
MOVIE 1
1150a
MOVIE M
1150b
CHAPTER LIST
1152
TRACK LIST
1151
TRACK 1
1153a
TRACK T
1153b
CLIP 1
1154a
CLIP N
1154b
PROPERTIES
1155
CHAPTER 1
1156a
CHAPTER C
1156b
END CMD
1158
MENU KEY
1157
MENUS LIST
1106
MENU 1
1160a
MENU N
1160b
BUTTON LIST
1161
BUTTON 1
1162a
BUTTON B
1162b
PROPERTIES
1163
CONNECTIONS SETS LIST
1107
CONNECT LIST 1
1170a
CONNECT LIST CL
1170b
CONNECT 1
1171a
CONNECT N
1171b
SOURCE 1
1172
TARGET 1
1173
1180
CHAPTER
1ST PLAY PGC
BUTTON
CHAPTER
BUTTON
TITLE

FIG. 11

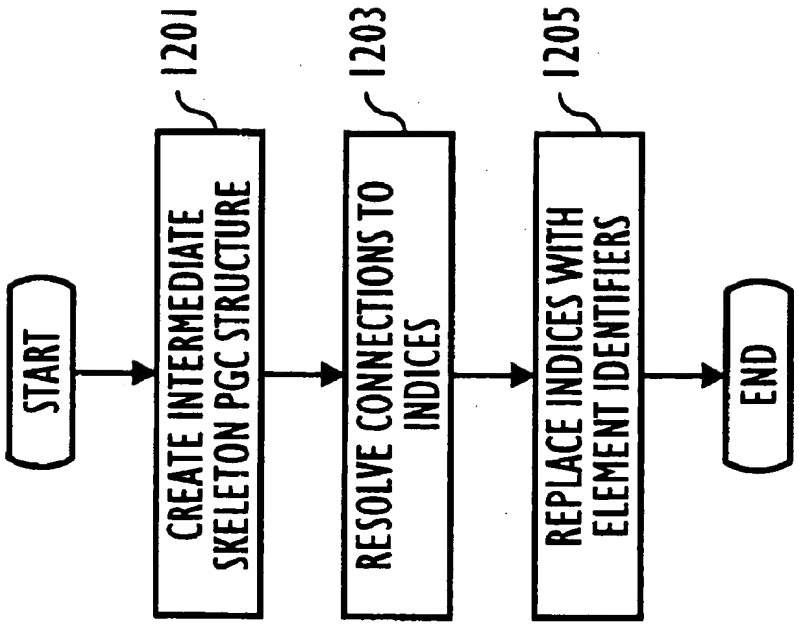

START
1201
CREATE INTERMEDIATE SKELETON PGC STRUCTURE
1203
RESOLVE CONNECTIONS TO INDICES
1205
REPLACE INDICES WITH ELEMENT IDENTIFIERS
END

FIG. 12A

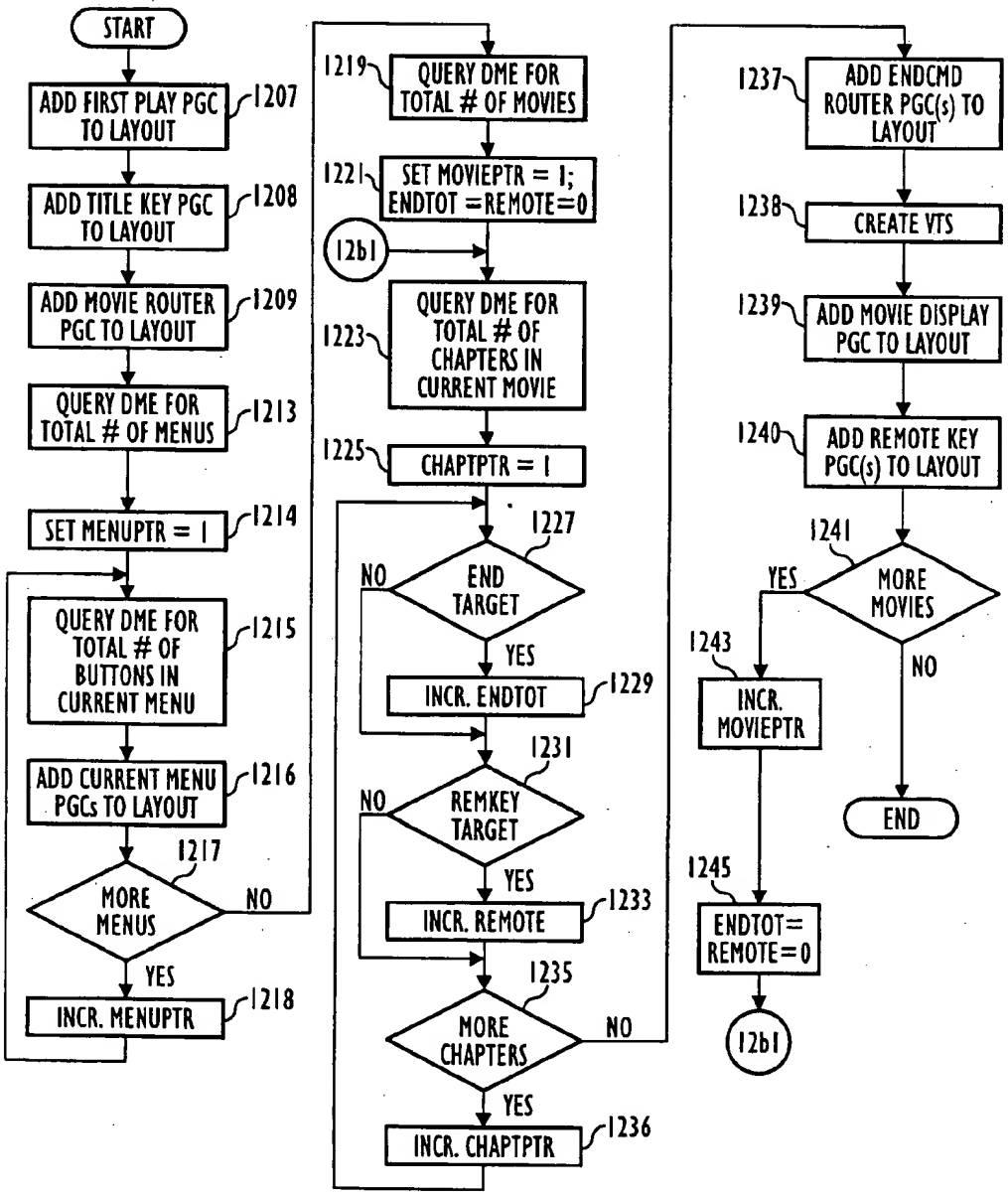

START
ADD FIRST PLAY PGC TO LAYOUT
1207
ADD TITLE KEY PGC TO LAYOUT
1208
ADD MOVIE ROUTER PGC TO LAYOUT
1209
QUERY DME FOR TOTAL # OF MENUS
1213
SET MENUPTR = 1
1214
QUERY DME FOR TOTAL # OF BUTTONS IN CURRENT MENU
1215
ADD CURRENT MENU PGCs TO LAYOUT
1216
1217
MORE MENUS
NO
YES
INCR. MENUPTR
1218
1219
QUERY DME FOR TOTAL # OF MOVIES
1221
SET MOVIEPTR = 1; ENDTOT =REMOTE=0
12b1
QUERY DME FOR TOTAL # OF CHAPTERS IN CURRENT MOVIE
1223
1225
CHAPTPTR = 1
1227
NO
END TARGET
YES
INCR. ENDTOT
1229
1231
NO
REMKEY TARGET
YES
INCR. REMOTE
1233
1235
MORE CHAPTERS
NO
YES
INCR. CHAPTPTR
1236
1237
ADD ENDCMD ROUTER PGC(s) TO LAYOUT
1238
CREATE VTS
1239
ADD MOVIE DISPLAY PGC TO LAYOUT
1240
ADD REMOTE KEY PGC(s) TO LAYOUT
1241
YES
MORE MOVIES
NO
1243
INCR. MOVIEPTR
END
1245
ENDTOT= REMOTE=0
12b1

FIG. 12B

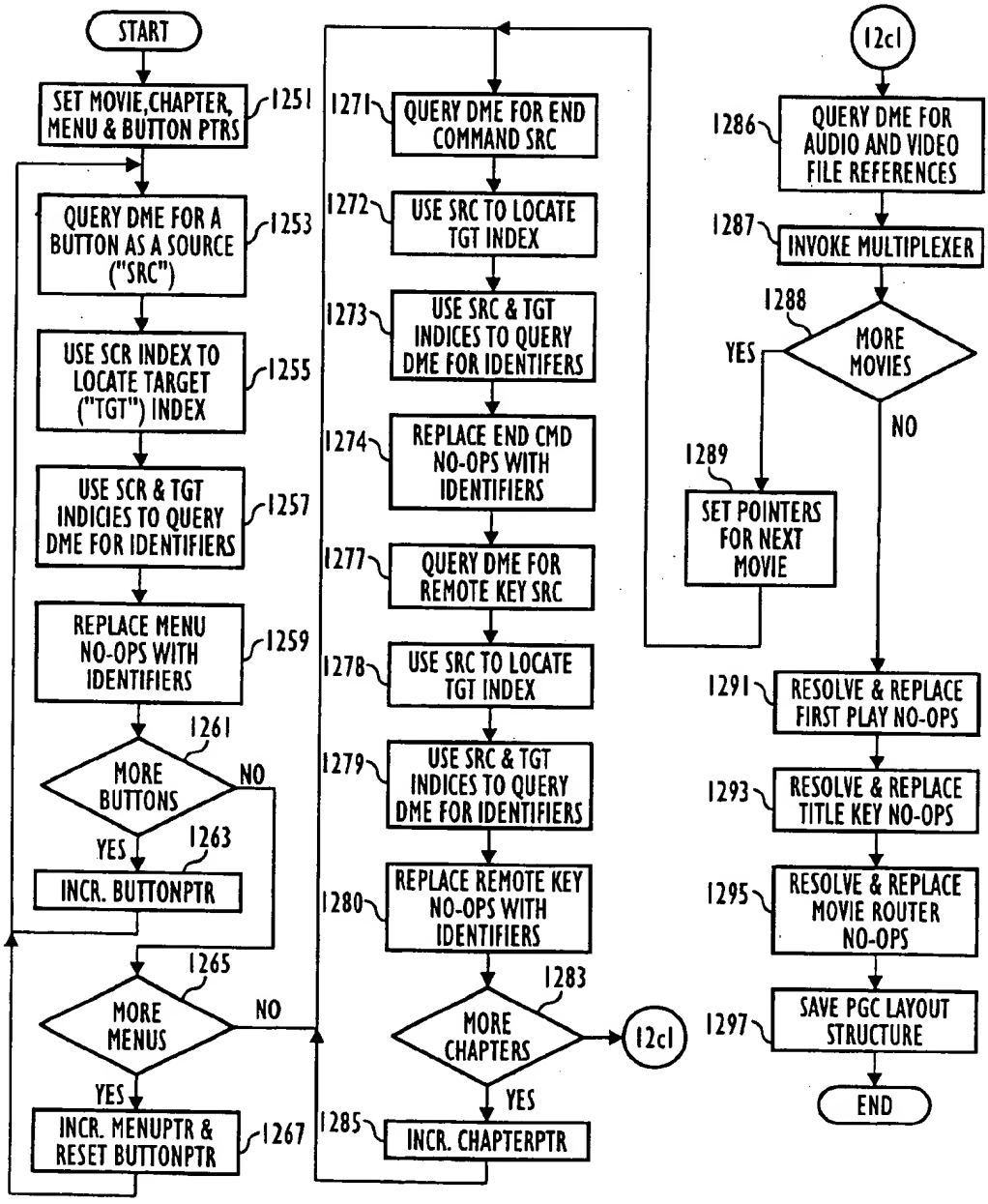

START
SET MOVIE,CHAPTER, MENU & BUTTON PTRS
1251
QUERY DME FOR A BUTTON AS A SOURCE ("SRC")
1253
USE SCR INDEX TO LOCATE TARGET ("TGT") INDEX
1255
USE SCR & TGT INDICIES TO QUERY DME FOR IDENTIFIERS
1257
REPLACE MENU NO-OPS WITH IDENTIFIERS
1259
1261
MORE BUTTONS
NO
YES
1263
INCR. BUTTONPTR
1265
MORE MENUS
NO
YES
INCR. MENUPTR & RESET BUTTONPTR
1267
1271
QUERY DME FOR END COMMAND SRC
1272
USE SRC TO LOCATE TGT INDEX
1273
USE SRC & TGT INDICES TO QUERY DME FOR IDENTIFERS
1274
REPLACE END CMD NO-OPS WITH IDENTIFIERS
1277
QUERY DME FOR REMOTE KEY SRC
1278
USE SRC TO LOCATE TGT INDEX
1279
USE SRC & TGT INDICES TO QUERY DME FOR IDENTIFIERS
1280
REPLACE REMOTE KEY NO-OPS WITH IDENTIFIERS
1283
MORE CHAPTERS
YES
1285
INCR. CHAPTERPTR
12c1
12c1
1286
QUERY DME FOR AUDIO AND VIDEO FILE REFERENCES
1287
INVOKE MULTIPLEXER
1288
MORE MOVIES
YES
NO
1289
SET POINTERS FOR NEXT MOVIE
1291
RESOLVE & REPLACE FIRST PLAY NO-OPS
1293
RESOLVE & REPLACE TITLE KEY NO-OPS
1295
RESOLVE & REPLACE MOVIE ROUTER NO-OPS
1297
SAVE PGC LAYOUT STRUCTURE
END

FIG. 12C

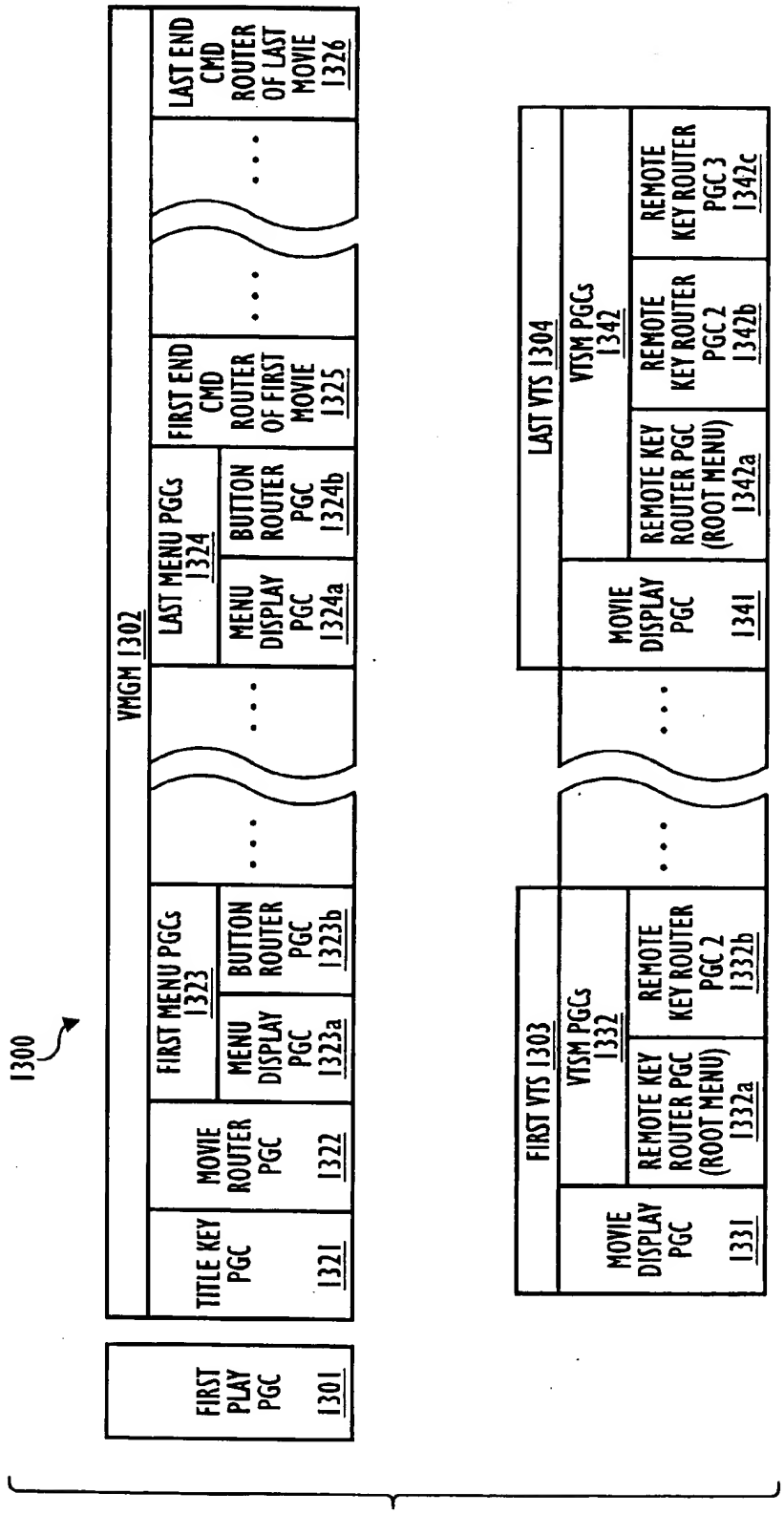

1300
FIRST PLAY PGC 1301
VMGM 1302
TITLE KEY PGC 1321
MOVIE ROUTER PGC 1322
FIRST MENU PGCs 1323
MENU DISPLAY PGC 1323a
BUTTON ROUTER PGC 1323b
LAST MENU PGCs 1324
MENU DISPLAY PGC 1324a
BUTTON ROUTER PGC 1324b
FIRST END CMD ROUTER OF FIRST MOVIE 1325
LAST END CMD ROUTER OF LAST MOVIE 1326
FIRST VTS 1303
MOVIE DISPLAY PGC 1331
VTSM PGCs 1332
REMOTE KEY ROUTER PGC (ROOT MENU) 1332a
REMOTE KEY ROUTER PGC 2 1332b
LAST VTS 1304
MOVIE DISPLAY PGC 1341
VTSM PGCs 1342
REMOTE KEY ROUTER PGC (ROOT MENU) 1342a
REMOTE KEY ROUTER PGC 2 1342b
REMOTE KEY ROUTER PGC 3 1342c

FIG. 13

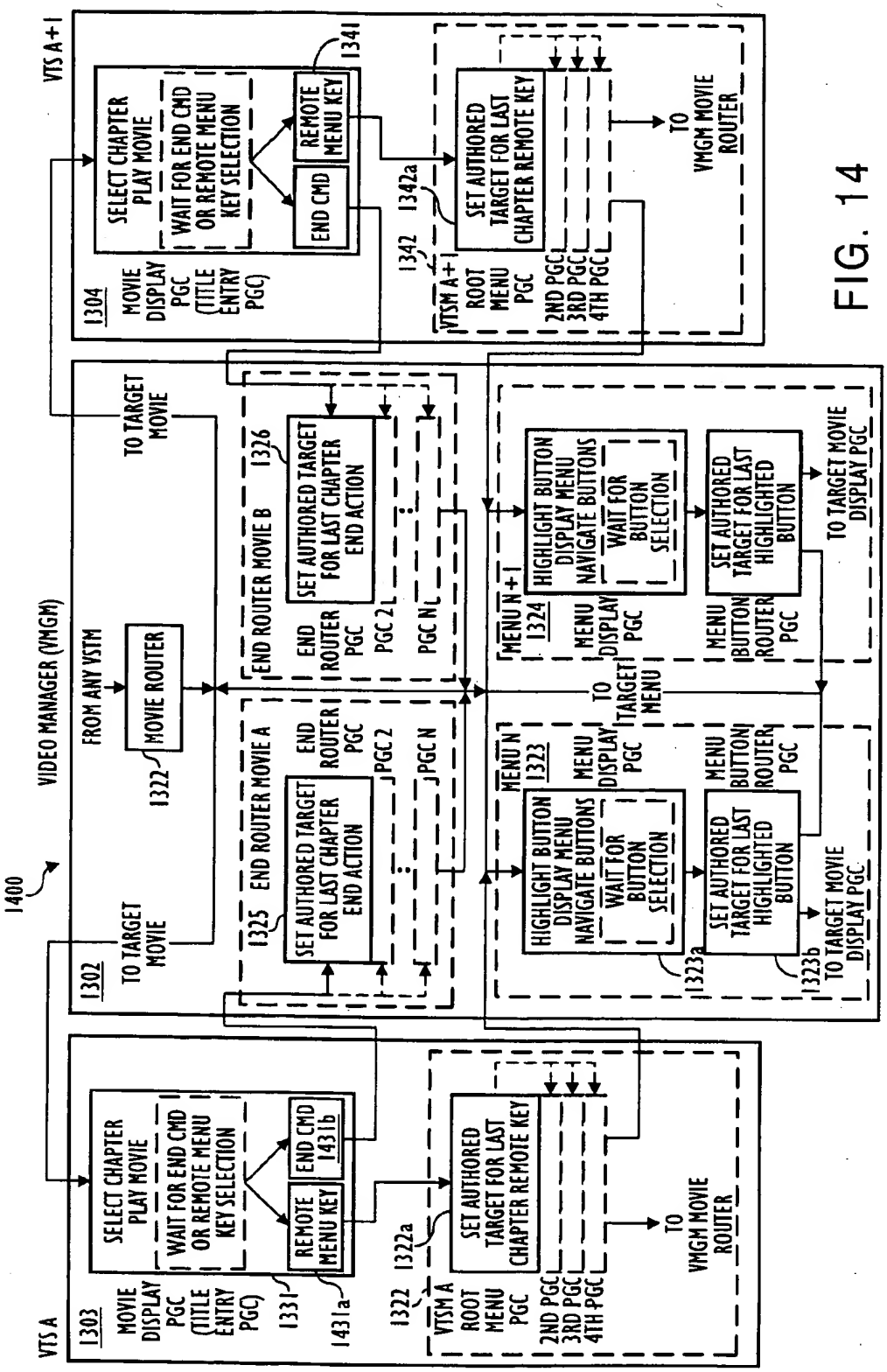

VTS A+1
SELECT CHAPTER PLAY MOVIE
WAIT FOR END CMD OR REMOTE MENU KEY SELECTION
REMOTE MENU KEY
END CMD
1341
1304
MOVIE DISPLAY PGC (TITLE ENTRY PGC)
1342
1342a
VTSM A+1 ROOT MENU PGC
SET AUTHORED TARGET FOR LAST CHAPTER REMOTE KEY
2ND PGC
3RD PGC
4TH PGC
TO VMGM MOVIE ROUTER
1400
VIDEO MANAGER (VMGM)
FROM ANY VSTM
MOVIE ROUTER
1322
TO TARGET MOVIE
1302
END ROUTER MOVIE B
1326
SET AUTHORED TARGET FOR LAST CHAPTER END ACTION
END ROUTER PGC
PGC 2
PGC N
END ROUTER MOVIE A
1325
MENU N+1
1324
HIGHLIGHT BUTTON DISPLAY MENU NAVIGATE BUTTONS
WAIT FOR BUTTON SELECTION
MENU DISPLAY PGC
SET AUTHORED TARGET FOR LAST HIGHLIGHTED BUTTON
MENU BUTTON ROUTER PGC
TO TARGET MOVIE DISPLAY PGC
TO TARGET MENU
MENU N
1323
1323a
1323b
VTS A
1303
1331
1431a
END CMD 1431b
1322a
VTSM A ROOT MENU PGC

FIG. 14

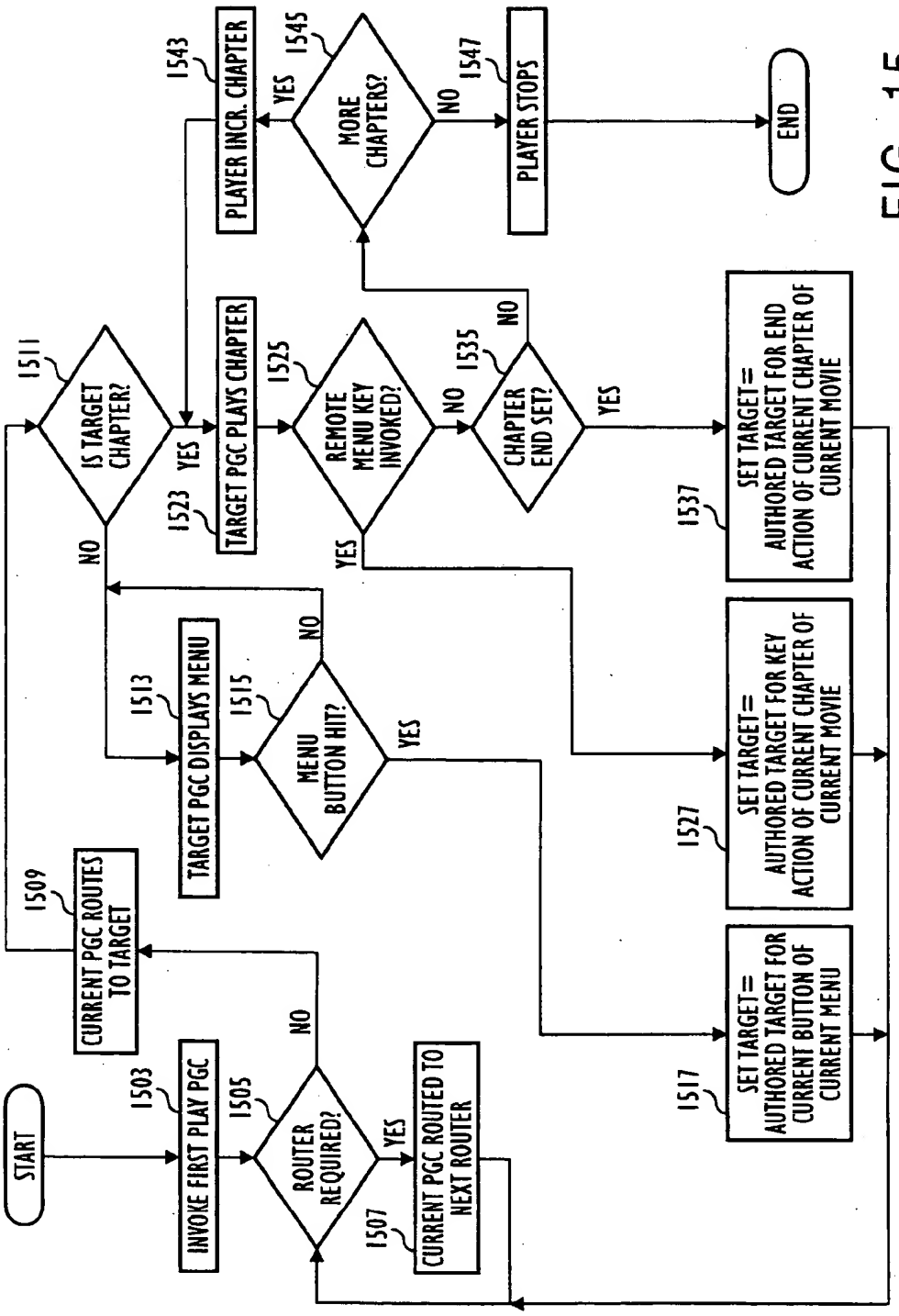

START
INVOKE FIRST PLAY PGC
1503
ROUTER REQUIRED?
1505
YES
NO
CURRENT PGC ROUTED TO NEXT ROUTER
1507
CURRENT PGC ROUTES TO TARGET
1509
IS TARGET CHAPTER?
1511
YES
NO
TARGET PGC DISPLAYS MENU
1513
MENU BUTTON HIT?
1515
YES
NO
TARGET PGC PLAYS CHAPTER
1523
REMOTE MENU KEY INVOKED?
1525
YES
NO
CHAPTER END SET?
1535
YES
NO
PLAYER INCR. CHAPTER
1543
MORE CHAPTERS?
1545
YES
NO
PLAYER STOPS
1547
END
SET TARGET= AUTHORED TARGET FOR CURRENT BUTTON OF CURRENT MENU
1517
SET TARGET= AUTHORED TARGET FOR KEY ACTION OF CURRENT CHAPTER OF CURRENT MOVIE
1527
SET TARGET= AUTHORED TARGET FOR END ACTION OF CURRENT CHAPTER OF CURRENT MOVIE
1537

FIG. 15

MENU AUTHORING SYSTEM AND METHOD FOR AUTOMATICALLY PERFORMING LOW-LEVEL DVD CONFIGURATION FUNCTIONS AND THEREBY EASE AN AUTHOR'S JOB

FIELD OF THE INVENTION

The present invention relates generally to mass data storage and retrieval, and more particularly to apparatus and methods for authoring a digital versatile disk.

BACKGROUND OF THE INVENTION

New mass data storage means provide not only for storing greater amounts of multimedia and other information, but also for more interactive data retrieval by consumers. For example, one such storage means is espoused by the "DVD Specification for Read-Only Disc, Physical, File Format and Video Specifications" (DVD Consortium 1997), hereinafter referred to as the "DVD Specification". Other examples include further DVD-related technologies (e.g. DVD-Audio, DVD-RAM, etc.) as well as non-DVD technologies.

The Physical and File System portions of the DVD Specification defines the physical encoding and organization of data for storage on read-only digital versatile disk ("DVD ROM") media. The Video portion of the DVD Specification defines a data set ("DVD-Video data set") with which pre-recorded DVD-Video discs must conform in order to assure proper reading, decoding and playback when inserted into a media reader/decoder ("DVD-player"). More specifically, the Video portion specifies how "control data" and audio/video "presentation data" are encoded and ordered within the data set. The control data determines how presentation of audio/video data will proceed when the disc is played back on a DVD-player and consists of low-level state information, data structures and instruction sets which govern what kinds of functions and user operations a DVD player can perform.

The DVD Specification is further hereby fully incorporated herein by reference as if repeated verbatim immediately hereinafter.

The process of encoding and authoring a DVD movie title, as currently practiced, includes a number of separate and distinct steps requiring similarly separate and distinct expertise. After movie production, raw film and/or video footage is edited, the soundtrack is edited and mixed, and a movie film or video master is created. This master is subsequently digitized, encoded as video and audio streams and stored as data files. In accordance with the DVD Specification, the Moving Pictures Expert Group ("MPEG-1 or MPEG-2") format is used to encode the video streams and any one or more of a number of specified formats (e.g. MPEG-1or MPEG-2 Audio, Dolby AC-3, PCM) is used to encode the audio streams. Graphic data (i.e. still or moving images for creating menus and other presentation data) is also created and stored in conventional graphic files. Finally, authoring guidelines, the encoded audio and video stream files and the graphic files are gathered for the authoring phase.

During authoring, a DVD author utilizes the guidelines and file information to construct a DVD movie-title. The authored movie-title determines what a user of a resultant movie title will see and hear, and what kinds of interactions the user can command when the movie title is played back by a DVD-player. The author organizes the video, audio and (often author-created) subtitle files, divides the movie into segments ("chapters"), creates menus, and specifies low-level instructions. The low-level instructions will set parameters, define fixed or optional jump points and their destinations and determine the order and options by which playback of still pictures, movie chapters and associated audio tracks will proceed based on the user's menu selections and/or use of other DVD-player controls (i.e. typically using a remote control device).

Once authored, the author's organizational decisions, subtitle, chapter and menu decisions, and low-level instructions are compiled into control data, and the encoded video, audio and subtitle streams, as well as the graphic data files, are multiplexed into presentation data, which together constitute the DVD-Video data set. Finally, this DVD-Video data is converted into a "disc image layout" file, which can be used to bum a "rite-once DVD-R" disc, or can be stored onto a tape to send to a DVD-ROM manufacturing plant for creating a "master" disc, which can then be used for replication.

Conventional DVD authoring systems comprise a computer system running an application-specific DVD authoring program. An exemplary, widely used conventional DVD authoring system is Scenarist-II.

Scenarist-II is essentially an attempted, nearly direct embodiment of the DVD Specification. Using Scenarist-II, an author organizes data streams, and constructs menus and DVD structures according to the DVD Specification. Top level structures (i.e. up to 99 "VTSs" and "VTSMs", a "VMG" and a "VMGM") are constructed by selecting the structure type and then populating the structure with one or more low-level command segments ("program chains" or "PGCs") including movie or menu references. Throughout this process, the author also selects from among available data formats, as well as from among the numerous DVD options and requisite parameters, using a number of provided lists and other data and parameter representations. Stated alternatively, all structures and PGC parameters, capabilities and references must be fully specified by the author on an ongoing basis during authoring.

Unfortunately, the DVD Specification is very complex, as are the conventional programs that attempt to embody it. Available options are extensive, as are the numerous listings of options and parameters within programs such as Scenarist-II. The potential combinations of structures and PGCs are also extensive, and many such combinations will not ultimately result in functional DVD movie-titles.

To make matters more difficult, the PGCs (i.e. basic and frequent constructs of the DVD Specification and therefore of programs such as Scenarist-II) are counter-intuitive. Often, many PGCs (including both operative and so-called "dummy" PGCs) must be used in specific combinations to provide a DVD consumer with even the most basic control capabilities. Limitations imposed by the DVD Specification must also be considered throughout the process. Thus, errors in planning and/or programming might well remain undetected until after a substantial number of structures are formed. In addition, given the sheer number of structures, PGCs, commands, options and parameters involved, identifying, locating and correcting errors is difficult and time-consuming.

Consequently, while providing extensive low-level control and an expedient authoring-to-compilation correspondence, conventional authoring systems require an extensive expertise with regard to both the DVD Specification and the authoring system itself. Further, even assuming such expertise, authoring is extremely time-consuming and is therefore typically very costly. In addition, even assuming

resolution of other factors, the time and expertise required would likely prevent authoring of even a preliminary movie-title as a directorial aid during the movie production process.

A further disadvantage of conventional authoring systems is that experimentation and all but necessary modification are often compromised due to time and cost considerations. Thus, many DVD movie titles (due to limited budget to support expensive authoring time) provide a DVD consumer with only minimal playback control, navigation flexibility and interactivity.

Accordingly, there is a need for an authoring system and method that enables DVD authoring in a manner removed from the structures and low-level instruction sets of the DVD Specification, thereby reducing the time, cost and complexity of the authoring process.

There is further a need for such an apparatus and method whereby authoring can be conducted in an intuitive manner, while maximizing flexibility and access to features provided by or otherwise not in conflict with the DVD Specification.

SUMMARY OF THE INVENTION

The present invention provides a data processing-system based authoring system and method that essentially removes an author from consideration of the structures and low-level instruction sets of the DVD Specification. More specifically, the present authoring system removes the ordered tasks associated with creating DVD structures and programming PGCs, and replaces them instead with an interactive, intuitive and graphical authoring environment.

The present invention further provides for flexible program flow in response to control events. Many interactive controls, menu button destinations and other features that are possible in accordance with the DVD Specification can be specified by an author in multiple instances and according to quick, intuitive and interactively modifiable selections. Thus the invention facilitates authoring of a DVD movie title by even an inexperienced author with context sensitive responsiveness to DVD consumer instructions and other DVD player-generated events.

Accordingly, a preferred embodiment of the present invention comprises an authoring engine having an integrated interface with which an author performs the above tasks a data management engine for storing and recalling authoring information, a simulator for viewing progressive and/or comparatively authored movie titles prior to compiling, a compiler, a multiplexer and an emulator for viewing authored movie titles after compiling and multiplexing.

Included within and facilitating the ability of these elements to remove an author from the DVD Specification are several abstractions. Preferably, the interface provides such "user abstractions" as arranging movies (i.e. data streams including video, audio, subtitles, chapter points and other elements), creating menu layouts (i.e. menus, menu buttons and still or moving images with or without sound) and specifying connections among these arrangements and layouts, each in a simple and intuitive, yet highly flexible way. Further abstractions include a network or connection-switching abstraction and a number of control and router PGC abstractions from which the connection-switching abstraction is constructed.

Authoring instructions entered through the interface are preferably broken down into component parts and stored by the data management engine. The invoked compiler, using only summary authoring information, preferably constructs a skeleton form PGC layout structure comprised of PGC abstractions corresponding to the number of authored movie elements. The compiler then completes the layout structure according to author-selected and default source-target connections.

Further according to a preferred embodiment, during playback of a resultant DVD movie title, a source PGC abstraction is invoked in response to DVD player and/or consumer instructions. The source PGC abstraction determines target information and transfers control, through necessary router PGC abstractions, to a target PGC abstraction. The target, in accordance with the target information, plays a movie chapter, displays a menu, or sets and/or modifies one or more DVD parameter.

These and other objects, advantages and benefits of the present invention will become apparent from the drawings and specification that follow.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is functional block diagram generally illustrating an authoring system according to a preferred embodiment of the invention;

FIG. **2** is a functional block diagram illustrating in more detail a preferred authoring program of the authoring system shown in FIG. **1**, according to the invention;

FIG. **3** is a screenshot of a preferred performance element arrangement interface portion of the FIG. **2** authoring program, according to the invention;

FIG. **4** is a blowup of the FIG. **3** screenshot showing, in more detail, a preferred authoring toolbar for accessing authoring program modules and functions;

FIG. **5** is a flowchart illustrating an exemplary method used by an author to create a performance element arrangement using the performance element arrangement interface portion of FIG. **3**;

FIG. **6***a* is a flowchart illustrating preferred responses of the authoring program to authoring while the performance element arrangement interface portion of FIG. **3** is active;

FIG. **6***b* is a flowchart further illustrating preferred responses of the authoring engine to authoring while the performance element arrangement interface portion of FIG. **3** is active;

FIG. **7** is a screenshot of a menu element layout interface portion of the FIG. **2** authoring program, according to the invention;

FIG. **8** is a flowchart illustrating an exemplary method used by an author to create a menu layout using the menu element layout interface portion of FIG. **7**;

FIG. **9** is a screenshot of a preferred connections interface portion of the FIG. **2** authoring program, according to the invention;

FIG. **10** is a screenshot of a preferred simulator interface portion of the FIG. **2** authoring program, according to the invention;

FIG. **11** is a functional block diagram of a preferred data management engine according to the invention;

FIG. **12***a* is a flowchart showing generally the operation of a preferred compiler according to the invention;

FIG. **12***b* is a flowchart showing how a compiler according to the invention preferably constructs a skeleton-form PGC layout structure;

FIG. **12***c* is a flowchart showing how the compiler preferably resolves source-target connections and substitutes those connections for null operations in a preferred skeleton-form PGC layout structure, according to the invention;

FIG. **13** is a block diagram showing the format of a preferred PGC layout structure according to the invention;

FIG. **14** is a functional block diagram showing a preferred connection-switching abstraction according to the invention;

FIG. **15** is a flowchart showing a preferred operation of the connection-switching abstraction of FIG. **14**, according to the invention;

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

For clarity sake, the discussed embodiment herein will be directed primarily toward storage according to the DVD Specification, and more specifically at authoring motion picture DVD ROMS ("movie titles"). It should be understood, however, that the present invention relates to a broad range of program and data storage and retrieval utilizing a variety of media, only a subset of which will be specifically identified herein. The types of DVD ROMS which can be authored are further in no way limited to movie titles. Other examples include but are not limited to music videos, documentaries, educational videos, corporate training, medical applications and other continuous play or interactive information which utilizes audio, video and/or other presentation data.

As illustrated in FIG. **1**, a preferred embodiment of authoring system **100** according to the invention preferably comprises electrically connected hardware elements including input devices **110**, processor **115**, memory **120**, storage **125**, MPEG encoder/decoder **130**, video I/O device **135** and audio I/O device **140**. Authoring system **100** further comprises software elements including operating system **150**, authoring engine **160**, data management engine **165**, compiler **170**, simulator **175**, emulator **180** and multiplexer **185**.

It will be apparent to those skilled in the art that several variations of the authoring system elements are contemplated and within the intended scope of the present invention. For example, given processor and computer performance variations and ongoing technological advancements, hardware elements such as MPEG encoder/decoder **130** may be embodied in software or in a combination of hardware and software. Similarly, software elements such as multiplexer **185** may be embodied in hardware or in a combination of hardware and software. Further, while connection to other computing devices is indicated as network I/O **145**, wired, wireless, modem and/or other connection or connections to other computing devices (including but not limited to local area networks, wide area networks and the internet) might be utilized. A further example is that the use of distributed processing, multiple site viewing, information forwarding, collaboration, remote information retrieval and merging, and related capabilities are each contemplated. Various operating systems and data processing systems can also be utilized, however at least a conventional multitasking operating system such as Windows95® or Windows NT® (trademarks of Microsoft, Inc.) running on an IBM® (trademark to International Business Machines) compatible computer is preferred and will be presumed for the discussion herein. Input devices **110** can comprise any number of devices and/or device types for inputting commands and/or data, including but not limited to a keyboard, mouse, and/or speech recognition. (The use of a keyboard and a mouse are exemplified throughout the discussion that follows.)

The FIG. **2** block diagram illustrates in greater functional detail an authoring program **201** of the preferred authoring system of FIG. **1**. As shown, authoring program **201** comprises authoring engine **160** (which includes interface **160***a*), data management engine **165**, compiler **170**, simulator **175**, emulator **180**, multiplexer **185**, output DVD data storage **290** and layout formatter **187**, user abstractions **285** and PGC abstractions **287**.

It is discovered through examination of the features supported by DVD players that the basic presentation data types and consumer controls available to an author of DVD movie titles can be generalized and then reconstructed as abstracted user data types and controls. Further, despite the complexity of the DVD Specification, many of its programming constructs can also be generalized and then reconstructed as abstracted DVD program chains ("PGCs") operating within a further abstracted network or connection-switching superstructure. Such user abstractions **285** and PGC abstractions **287**, as integrated into authoring engine **160**, data management engine **165** and compiler **170** (as illustrated), effectively remove an author using authoring program **201** from consideration of DVD Specification **205**. These abstractions further remove such consideration without unduly limiting, for most practical purposes, authoring flexibility, PGC efficiency or interactive responsiveness of a resultant DVD-ROM, among other factors. In addition, these abstractions provide a framework of re-useable components that are readily adaptable to further modification for providing improvements, and for re-use in a variety other DVD and non-DVD applications.

Authoring program **201** is preferably implemented in C++, an object-oriented language, for reliability, updateability and other known generalized advantages of object-oriented programming. Those skilled in the computer arts will appreciate however, that despite such advantages, other environments and/or programming languages of various object-oriented and non-object-oriented types can also be utilized.

Operationally, an author enters authoring information and instructions for activating and controlling authoring program **201** through interface portion **160***a* of authoring engine **160**. Authoring engine **160** interactively receives entered information and commands by correspondingly adjusting interface portion **160***a*, invoking a further authoring program module, sending entered authoring information to data management engine **165**, retrieving authored information from data management engine **165**, and sending and/or retrieving presentation data from presentation data storage **203**. Data management engine **165** responds to authoring engine **160** by receiving and storing authored information from authoring engine **160** and/or sending information, which it retrieves from storage (and/or from a remote source), to authoring engine **160**. Simulator **175** responds to authoring engine **160** by retrieving authoring data from data management engine **165**, retrieving multiplexed presentation data from multiplexer, and simulating an authored DVD-ROM in conjunction with interface **160***a*.

Compiler **170** responds to authoring engine **160** by retrieving authored information from data management engine **165**, compiling the information and storing the compiled information (".ifo files") in output DVD data storage **290**. Emulator **180** responds to authoring engine **160** by retrieving compiled data from output DVD data storage **290**, retrieving multiplexed data from output DVD data storage **290** and emulating an authored DVD-ROM in conjunction with interface **160***a*. Multiplexer **185** responds to authoring engine **160** by receiving DVD parameter information from compiler **170**, retrieving presentation data from presentation data storage **203** and combining the retrieved information and data in accordance DVD Specification **205**. Multiplexer **185** then stores the combined information and

data ("DVD data stream" or ".vob file") in output DVD data storage **290**. Layout formatter **187** retrieves the .vob files and ifo files from output DVD data storage **290** and combines these files into a single "disc image" file, which it then stores in disc image file storage **207**. The disc image file can then be sent through network I/O **145** (FIG. **1**) to additional apparatus for further review, processing and/or for burning one or more DVD-ROMs **207**.

FIGS. **3** through **10**, with reference to FIG. **2**, illustrate how an interface according to the invention enables an author to assemble a movie title essentially removed from DVD programming specifications **207** (FIG. **2**) of the DVD Specification. Preferred interface **160***a* is illustrated as an application running under a Windows95® or Windows NT® (trademark of Microsoft, Corp.) operating system.

The FIG. **3** screenshot illustrates a preferred authoring window **300**, which an author can utilize to select an arrangement of audio-visual material including video segments ("video clips"), audio segments ("audio clips") and subtitles (hereinafter referred to collectively as "performance data").

Authoring window **300** is divided into movable, modifiable and replaceable groupings or "views" and "panels" including presentation data panel **301**, performance assembly panel **302**, assembled elements panel **307**, log panel **308** and preview video panel **309**. Assembly panel **302** is further divided into video assembly portion **320**, audio assembly portion **330** and subtitle assembly portion **340** (which are collectively referred to herein as performance view **303**), and performance tools portion **360**. Authoring window **300** also includes authoring toolbar **399***a* and menu bar **399***b*. For clarity sake, the following discussion assumes that a single, continuous movie is being authored (i.e. a movie having component video, audio and subtitle data streams each of which begins at the start of the movie and ends at the conclusion of the movie).

Presentation data panel **301** provides a display listing for each presentation data file that an author has selected and loaded for use in assembling movies and menus either during a current authoring session or when continuing a re-initiated, prior authoring session. File listings include file name **311**, file duration **313**, and file type **315** parameters. File name **311** lists the name of a file. File duration **313** lists the playback duration of files such as video data files and audio data files. File type **315** alternatively lists a file format, which is generally indicated by a filename extension, or a recognized data type such as "video" data or "audio" data. As will be further discussed, presentation data file listings can be used interactively during an authoring session.

Performance assembly view **303** of performance assembly panel **302** is used by an author to graphically and interactively assemble loaded video and/or audio data, to add and assemble subtitles, and/or to add chapter points. For these purposes, performance view **303** includes video assembly portion **320**, audio assembly portion **330**, subtitle assembly portion **340** and chapter assembly portion **350** respectively. Video assembly portion **320** is used by an author to assemble graphic objects referencing stored video data files ("video clips"). As discussed, these files, once initially selected, are listed in presentation data panel **301**. Video frame thumbnails **323***a* and **323***b* are indicative of chapter points as will be further discussed herein.

Audio assembly portion **330** of performance assembly panel **302** is used by an author to receive graphic objects referencing stored audio data files ("audio clips"). As with video clips, audio clips, once selected for use, are listed in and selected from presentation data panel **301** for arrangement purposes. Up to eight (alternate language) audio data streams or audio "tracks", exemplified by audio tracks **331***a* through **331***c*, are available in accordance with DVD Specification **205** (FIG. **2**). Audio bars **332***a* and **332***b*, which represent author-arranged audio clips, have a length that reflects the playback time of the audio data represented. Separators **333** are further indicators of chapter points, as with video frame thumbnails **323***a* and **323***b* of video assembly portion **320**. Audio tracks **332***a* through **332***c* further include audio encoding indicators **334***a*, audio format indicators **334***b*, track number indicators **335** and selected language indicators **336**, which are indicative respectively of audio data file encoding and playback format, selectable audio track number **336** and modifiable language label **335**. Language labels **335** can be set by author selection or, as is expected, automatically by recognition of languages spoken in a recorded dialog of a respective audio track.

Subtitle assembly portion **340** provides for entry, retrieval and/or editing of up to thirty-two (alternate language) frame-based subtitle sequences, as exemplified by tracks **341***a* and **341***b*. Exemplary subtitle frames **342***a* and **342***b* illustrate textual subtitle contents. Subtitles are entered in a conventional manner using a conventional text editor (not shown) which is invoked by activating a subtitle frame (e.g. by menu selection or double-clicking) and/or by retrieving a pre-existing subtitle file using, for example, presentation data panel **301**. As with audio assembly portion **330**, subtitle portion **340** includes selectable track numbers and modifiable language label indicators.

Performance assembly view **303** also includes chapter assembly portion **350**, which is used by an author to graphically and interactively assemble chapter points. Chapter assembly portion **350** includes wall clock **351**, reference offset clock **352**, author-assembled chapter indicators **353***a* through **353***c*, chapter time indicators **354***a* through **354***c* and reference time indicators **355***a* through **355***c*. Wall clock **351** indicates a time within a video clip corresponding to a cursor position over chapter portion **350** of assembly panel **302**. Offset clock **352** indicates the start time of a currently indicated video clip according to the reference timecode of a master tape (i.e. from which the video data file was created). Chapter indicators **353***a* through **353***c* show chapter points (i.e. points to which a DVD-ROM consumer can advance) as arranged during authoring. Chapter time indicators **354***a* through **354***c* and reference time indicators **355***a* through **355***c* display the elapsed time of corresponding selected chapter points from the start of a movie and from the start of a clip respectively. Reference times are typically recorded (and thus can be selectively retrieved and displayed) utilizing Society of Motion Pictures and Television ("SMPTE") timecode.

As noted earlier, performance assembly panel **302** and the other panels and views of authoring window **300** are replaceable. Tabs **302***a* provide one alternative control structure for selectively switching between initiated or "open" authoring tasks, for example, to alternate between assembling presentation data of multiple movies, for creating menu layouts, and/or for other authoring tasks. Other control structures include menu options (not shown) for selectively de-coupling panels and transport enabling controls (**362***a* through **362***c* and **363***a* through **363***b*), and further for re-coupling in the illustrated default arrangement, in an author-selectable arrangement and/or interactively by an author. Panels can be resized and/or re-arranged among other window capabilities, as will be understood by those skilled in the art in view of the discussion herein.

Assembly tools portion **360** of performance assembly panel **302** comprises selectable zoom controls **361***a* through **361***c*, preview transport buttons including stop **362***a*, play **362***b* and frame advance **362***c*, preview transport start time selector **363***a* and stop time selector **363***b*, selected clip indicator **364***a* and total clips indicator **364***b*. Zoom controls **361***a* through **361***c* are used respectively for increasing the viewable data range of a selected area within performance assembly view **303** of performance assembly panel **302**, for selecting a portion of performance assembly view **302** for such viewing, and for decreasing the viewable data range. Transport controls **362***a* through **362***c* provide video playback control when previewing a video clip, audio clip and/or subtitle data using preview video panel **309**, or when selecting a representative video frame in a video clip as a preview thumbnail (as with exemplary thumbnails **323***a* and **323***b*). Transport control **362***a* halts video, audio and/or subtitle playback, transport control **362***b* initiates/continues playback and transport control **362***c* provides for per-frame ("step") viewing, as will be understood by those skilled in the art. Start and end time selectors **363***a* and **363***b* are used respectively for selecting and monitoring video, audio and/or subtitle playback position and for setting and monitoring a playback stop time.

Assembled elements panel **306** provides interactive and selectable listings of authored contents of a current movie title, including but not limited to movie volume **361**, movies **362** and menus **363**.

Log panel **308** provides selectable progress reports and other information relating to decoding/encoding of presentation data, compiling and layout of a disk file format according to DVD disk format specifications **205** (FIG. **2**). These reports are automatically created and can be accessed using log tabs exemplified by tabs **381** and **383**.

Preview video panel **303** selectively displays a video frame corresponding to a cursor position over assembly panel chapter portion **350**, video assembly portion **320**, audio assembly portion **330**, subtitle portion **340** and/or chapter portion **350** of assembly panel **302**. In addition, preview video panel is used for previewing video data using transport controls **362***a* through **362***c*, start and stop time selectors **363***a* and **363***b* or directly invoking the panel using selection or drag-and-drop capabilities. (As will be understood by those skilled in the art, encoded video and audio files are decoded and buffered, as needed, for playback in a conventional manner using MPEG encoder/decoder **130** of FIG. **1**.)

The following toolbar chart lists the respective elements of toolbar **399**. It will be understood by those skilled in the art, in view of the discussion herein, that the toolbar elements can vary substantially and includes user-defined expandable and replaceable elements. The elements shown are provided as defaults.

Label	Referenced as	Description
401	New volume	Loads default values and adjusts the interface for a new movie title.
403	New menu	Loads default values and adjusts the interface for a new menu layout.
405	New movie	Loads default values and adjusts the interface for authoring a new movie.
407	Connections	Switches to an existing connections interface or adjusts the interface, according to default values for initially setting connections.
413–415	Cut, copy and paste	Provide conventional functions except as described herein for connections.
421	Compile start	Initiating compiler operation.
423	Compiler stop	Interrupts compiler operation.
425	DVD Layout	Invokes DVD Disk layout operation.
427	Write Tape	Provides for output of multiplexed data stream to tape.
429	Simulator	Invokes simulator

The FIG. **5** flowchart illustrates, by way of example and with reference to FIGS. **3** and **4**, how an interface in accordance with the invention enables an author to assemble performance data and objects without consideration for structures, commands or ordered tasks imposed by DVD programming specifications **207** (FIG. **2**). Select, open and drag-and-drop, among other operations, and clicking, double-clicking, click-and-drag and other user actions associated with graphic interfaces are well known and will not be further expounded upon herein.

As shown, in step **505**, an author initiates a new project ("volume") by selecting new volume **401** (FIG. **4**). In step **510**, the author initiates a new movie by selecting new movie **405**. In step **515**, the author adds video and audio files to presentation data panel **301** (FIG. **3**) for potential use in the volume by movies and menus. In step **520**, the author can preview a video file in preview panel **304** by dragging its icon in presentation data panel **301** to preview panel **304** and/or, if desired, by invoking transport controls **362***a* through **362***c*, preview timer **393** and/or other playback-related controls. In step **525**, the author adds a selected video clip to the currently opened movie by double-clicking its icon in presentation data panel **301** or by dragging the icon from presentation data panel **301** to video assembly portion **320** of performance view **303**. In step **530**, the author can select a video frame thumbnail other than a first frame for reference viewing by dragging the pointer of thumbnail timer **325***a* and/or by using transport controls **362***a* through **362***b*.

In step **535**, the author can preview an audio file by selecting its icon in presentation data panel **301** and using controls including stop **362**, play **362***b*, using start time and end time selectors **363***a* and **363***b* and/or using other play-related controls. In step **540**, the author adds a selected audio clip to a next available track of the currently opened movie by double-clicking its icon in presentation data panel **301**. (Alternatively, the author can add a selected audio clip to a specific audio track by dragging the icon from presentation data panel **301** to a selected track in audio assembly portion **330** of performance view **303**. In step **545**, the author selects a language label by selecting selected language indicator **335** and selecting a listed element.

In step **550**, the author opens a subtitle frame and enters subtitle information for display in a video frame during playback of video clips. In step **555**, the author selects a language label corresponding to the subtitle track containing the subtitle frame. If, in step **560**, the author elects to add more performance data, then the author returns to step **520**.

In step **565**, the author moves a cursor within chapter assembly portion **350** of performance view **303** to view video frames available as chapter points. In step **570**, the author selects a chapter point. If, in step **575**, the author elects to add more chapter points, then the author continues at step **565**.

In step **580**, the author selects an audio track number and optionally selects a subtitle track number and/or playback start and/or end times before selecting play button **362***b* to preview playback of the video clip and the audio clip referenced by the selected track number.

The FIGS. **6***a* and **6***b* flowchart (with reference to FIGS. **2** and **3**) generally illustrates responses by the preferred authoring program **201** to an author's actions according to the invention. As shown, if in step **602** an author selects a movie assembled in a prior authoring session, then, in step **604**, data management engine **165** (FIG. **2**) loads related parameters and, in step **606**, sends the parameters to authoring engine **160**. Otherwise, default parameters for a new movie are loaded in step **608**.

In step **609**, authoring engine **160** updates assembled elements panel **307** (FIG. **3**) and other affected interface **160***a* elements to indicate the movie parameters. If, in step **612**, the author selects presentation data files, then data management engine **165** loads and sends the respective presentation data file parameters to authoring engine **160** in step **614**, which updates presentation data panel **301** in step **616**. If, in step **622**, the author assembles one of the selected video clips, then authoring engine **160** accordingly updates video assembly portion **320**, chapter assembly portion **350** and offset clock **352** in step **624**, updates assembled elements panel **307** in step **626**, and sends the video clip parameters to data management engine **165** for storage in step **628**. Similarly, if the author assembles one of the selected audio clips in step **632**, then authoring engine **160** updates the selected track of audio assembly portion **320** in step **634**, updates assembled elements panel **307** in step **636**, and sends the audio clip parameters to data management engine **165** in step **638**. If, in step **642**, the author assembles subtitle data, then authoring engine **165** updates subtitle assembly portion **340** in step **644**, updates assembled elements **307** in step **646**, and sends subtitle data and parameters to data management engine **160** in step **628**.

If, in step **652**, the author moves an interface **160***a* pointer (e.g. a mouse pointer) within chapter assembly portion **360**, then in step **654** authoring engine **160** updates wallclock **351**, finds an I-frame (i.e. a video frame that is completely described without reference to other frames) within the video clip corresponding to the mouse pointer position and displays the I-frame in preview video panel **309**. If, in step **672**, the author assembles a chapter point, then authoring engine **160** updates video assembly portion **340** and chapter assembly portion **350** in step **674**, updates assembled elements panel **307** in step **676**, and sends corresponding chapter parameters to data management engine **165** in step **678**.

The FIG. **7** screenshot illustrates the preferred authoring window **300** of FIG. **3** with the performance data assembly panels replaced by panels for allowing an author to layout menus. More particularly, menu layout panel **701** and menu tools panel **702** are selected, sized and positioned to replace performance view **303** of FIG. **3**. An exemplary menu layout including graphic and textural images is shown in menu layout panel **701** for purposes of illustration. Menu layout panel **701** is used visually and interactively by an author to retrieve, add, place and modify menu elements using menu tools panel **702** selections.

In accordance with the DVD Specification, menu elements presentable to a DVD consumer can include a background image ("background"), an overlay image ("subpicture") and up to twenty-five buttons. For the present example, author-selected background **710** is a multicolor design, and author-selected subpicture **711** includes the textural information, Dolby Demo 1, Dolby Demo 2, Play Both Demos and Main Menu. Four author-created buttons **720***a* through **720***d* including button frames **721***a* through **721***d* are also shown. Each of button numbers **722***a* through **722***d* is added by authoring program **201** (FIG. **2**) in response to creation of a respective button for identification purposes (i.e. during authoring and for use in compilation).

Menu tools panel **702** comprises controls for implementing selectable menu element parameters and for selectably altering the display characteristics of elements within menu layout panel **701** during an authoring session. For example, color selection boxes **732**, **734**, **736** and **738** allow an author to choose a button outline color for display (in a consumer viewing scenario) when a button is not selected ("normal"), when a consumer points at the button ("selection") and when a button is invoked ("action") respectively. An author can also select the opacity of the buttons for these cases using opacity sliders **733**, **735**, and **737** respectively. Similarly, an author can select button shapes and other characteristics by selecting one of the layout feature tabs **739** and utilizing the tool sets that appear in a respective tool set panel (not shown). An author might, for example, utilize prior button shape, color, texture, opacity and/or normal, selection and activation color combinations used with a prior authoring session as either a starting point for further changes or without further modification. Other parameter combinations might also be utilized. Safe area toggle **755***a* allows an author to selectively display safe area indicator **755***b* of menu layout panel **701** (which bounds an area that is assured to be displayed on a consumer television). Display controls **751** and **752** provide for altering the characteristics indicated which, in light of the prior discussion, will be understood by those skilled in the art without further edification.

Layout feature tabs **749** also provide access to button ordering tools (not shown). As with other authoring parameters, an author can selectively utilize an existing order of buttons that will be traversed in a currently displayed menu when a consumer pushes directional buttons on a remote control device. An alternative order can also be set using any number of methods including but not limited to using a displayed remote control device or dragging an arrow from a starting point to an ending point. Such features and their operational characteristics, given the foregoing, will be understood by those skilled in the art without further edification.

The FIG. **8** flowchart shows how the actions required for laying out a menu are consistent with those for assembling performance data. Once again, authoring is visually and interactively achieved without requiring any specific ordering of actions. Therefore, as with performance data assembly, the specific ordering of actions is given for purposes of illustration only.

As shown, in step **805**, the author selects background and subpicture files for inclusion in a menu layout. Selected files will appear in presentation data panel **301** (FIG. **7**). In step **810**, an author adds a background and a subpicture to the current menu by double-clicking on file listings, dragging the files to menu layout panel **701** or by using a similar method. In step **815**, the author draws (i.e. drags a box) around subpicture text forming a button frame, thereby indicating button placement directly in menu layout panel **701**. If, in step **820**, more button frames remain to be added, then the author returns to step **815**.

In step **825**, the author selects a button and sets shape, size, opacity and other parameters using preset combinations

and/or color selection boxes **732**, **734**, **736** and **738**, opacity sliders **733**, **735**, and **737** and/or other tools. In step **830**, the author sets the intra-menu button order in the manner already described. If, in step **840**, more menus remain to be created, then the author selects add menu button **413** in step **840**, and returns to step **805**. New elements appear in assembled elements panel **307** and control data (i.e. relating to added elements and their layout characteristics) are sent to data management engine **165** (FIG. **2**) as with performance data assembly.

The FIG. **9** screenshot illustrates a further selectable configuration of the FIG. **3** interface for linking together presentation data, menu layouts, buttons within menu layouts and available control functions of a DVD player. As shown, connection view **901** includes available targets panel **903** and linking panel **905**. Linking panel **905** further includes available sources portion **950** and connected targets portion **960**. While connections view **901** is active, assembled elements panel **307** can further be used as a selection means for navigating more quickly to a desired target within available targets panel **903**.

Operationally, an author forms a link or "available connection" simply by copying (i.e. performing a copy action or dragging) a target from available targets portion **903** to a position in connected targets portion **960** that is in the same row as a desired source in available sources view **950**. As with assembling a movie and menu layouts, an author can interactively remove, move or otherwise modify links in a conventional manner. For example, a link can be removed by deletion or a target can be moved or copied to another row in linking portion **905**.

As with arranging performance data and forming menu layouts, an author has easy and complete flexibility in adding interactivity to a consumer's viewing experience. A DVD movie can be authored, for example, such that entry and exit from a menu can be controlled by any available event. Referring also to the FIG. **10** simulator window **1000**, any menu button can further be linked to any DVD event, including but not limited to a chapter point (e.g. chapter point **953**), the end of chapter playback or depressing a DVD remote control device menu button **1020** and **1040** (FIG. **10**). A particular menu button can also be used as a target in multiple instances, as might be creatively appropriate.

Thus, for example, a consumer interface can be quickly and easily created which is interactively responsive ("context sensitive") to a consumer's actions. Stated alternatively, an interface can be authored such that, for example. the conclusion of a specific chapter playback or menu button activation will determine a next chapter playback, a next menu or even a next menu wherein an author-selected menu button is highlighted.

Among the reasons for such ease and flexibility is that, contrary to conventionally authored DVD movies, program chains are not created during the authoring process. Similarly, connections specified during authoring are not permanent ("hard wired"). Rather, program chains are not created until compilation and available connections are not fully resolved until playback, each according to additional abstractions of the invention, as will be further discussed herein.

The FIG. **11** block diagram illustrates the structure of a preferred data management engine **165** (FIG. 1) according to the invention. As illustrated, data management engine **165** only partially reflects the interface constructs and the structures of the DVD Specification. While reflecting interface abstractions (e.g. a movie, menu and connection based movie-title description) and DVD Specification requirements (e.g. first play jump source), data management engine **165** is further structured as a flexible network of data storage and distribution objects that also reflects other abstractions of the invention.

One further abstraction, for example, is a model of a DVD player, a consumer's controller and the compiled authoring instructions as an actively connection-switched network. Within this network, DVD program chains representative of action-oriented authoring instructions ("routers"), perform switching among available connections in response to DVD-player (i.e. consumer) instructions, thereby re-directing program flow and control. Control-receiving program chains then perform more localized tasks (e.g. such as displaying a menu). Stated alternatively, a router program chain resolves an available connection from a DVD-player control instruction to a receiving program chain, which again routes control or executes the instruction. Further abstractions also include models of program chains for performing a common base functionality in a same or similar manner using a derived common program chain structure.

Such an arrangement provides real world flexibility and efficiency. For example, data management engine **165** supports authoring flexibility with regard to source-target connections that are switchable. Further, given the power of even conventional computer systems, data management engine **165** is sufficiently robust to enable the interactive operation of interface **160***a* (FIG. **2**) as well as minimal compilation times of compiler **170** (i.e. only milliseconds) without direct interface or DVD program specification **205** correlation. Data management engine **165** is therefore also readily adaptable to interface variations and further interfaces, as well as to compiler variations and other compilers supporting other DVD and non-DVD data storage and/or retrieval applications.

Referring again to FIG. **11** and with further reference to FIG. **2**, data management engine **165** comprises a root volume object **1100**, which manages data management engine **165** communication and storage. Volume object **1100** provides an interface for communicating messaged data to and from its component parts, including title key jump source **1101**, first play jump source **1102**, media database **1103**, DVD layout properties **1104**, movies list **1105**, menus list **1106** and connections list **1107** (objects). Media database **1103** further includes media files list **1130**, which stores pointers to media files referred to by the performance data arrangement as a result of authoring.

In addition, each of the presentation data objects (i.e. movies list **1105** and menus list **1106**) and a connection sets list object **1107** contain links to other data management engine objects in the form of an object tree. More specifically, movies list **1105** is linked to movie objects movie-1 **1150***a* through movie-M **1150***b*, wherein M is the total number of movies authored for storage on a single DVD-ROM ("movie title"). Each movie object contains a respective track list object **1151** and a respective chapter list object **1152**. Each track list object **1151** contains respective track objects, track- **1153***a* through track-T **1153***b*, wherein T is the total number of tracks authored within a respective movie. Track-1 through track-T further contain clip lists, which in turn contain clip objects clip-1 **1154***a* through clip-CL **1154***b* (and wherein CL is the total number of clips in a given track within a given movie). Finally, each clip object contains a respective clip properties object, as exemplified by clip object **1155**.

Menu objects are structured in a manner similar to that of movie objects. Menus list object **1160** contains menu objects

menu-1 **1160***a* through menu-N, wherein N is the total number of menus authored for storage on a given DVD-ROM. Each menu object further contains a respective button list object (e.g. object **1161**), each button list object contains a respective button objects (button-**1** **1162***a* through button-B I **162***b*) and each button object is linked to a button properties object (e.g. object **1163**). B indicates a total number of buttons in a respective menu.

Finally, connections sets list **1107** contains respective connections lists (i.e. connect-list-**1** **1170***a* through connect-list-CL **1170***b*), wherein CL is the total number of connections lists authored for storage on a given DVD-ROM. Each connect-list is further linked to respective connections objects (i.e. connect-1 **1171***a* through connect-CN), wherein CN is the total number of connections authored to facilitate flexible program flow and control. Each connections object (**1171***a* through **1171***b*) represents an action-oriented switch between a respective source and a respective target (as indicated by source-pointer variable **1172** and target-pointer variable **1173**), as will be discussed further herein.

Where applicable, each object includes an indexed object list having a pointer to each connected dependent object (i.e. an object "further down the tree" as illustrated), as well as a totals variable. The object list is updated to include new dependent objects as these objects are created ("instantiated") to reflect, for example, an added chapter point or menu. Dependent objects are similarly removed from the object list according to authoring deletions. Totals variables are also updated during authoring to reflect each corresponding dependent object instantiation and deletion. Undo and redo operations are handled in a conventional manner using authoring instructions which are further conventionally stored within respective objects during each authoring session.

Using this structure, data management engine **165** breaks down or filters control data generated during authoring into its basic component parts for storage in a corresponding object's indexed data list. These basic component parts are then retrieved by authoring engine **160**, or retrieved and reconstructed into an applicable form by compiler **170**, as needed.

Operationally, data management engine **165** receives messages from authoring engine **160** in response to and reflecting each author modification of a performance assembly, menu layout or connection. Volume **1100** receives the message, polls its contained-objects list for a recipient object according to the message type, and sends the message to the matching recipient object. If the message includes a reference to a title key source or a first play source (which is author-slectable in connections view **901**), then volume **1100** sends the message respectively to either title key jump source **1101** or first play jump source **1102**. Upon receipt, title key jump source **1101** or first play jump source **1102** will accordingly store included data, delete stored data or modify stored data.

If a received message includes a reference to a video, audio or subtitle file, then volume **1100** sends the message to media database **1103**. If the message contains an instruction to add a data element, then media database **1103** stores the data (which will include a pointer to a media file) in media files list **1130**. If the message contains an instruction to delete a stored pointer, then media database **1103** deletes the pointer. If the message contains an instruction to modify a stored pointer (e.g. if the file was moved to a new location), then media database **1103** locates and replaces the file pointer. Media database **1103** further updates its totals variable to reflect additions and deletions.

If a received message type relates to the content of a movie arrangement, menu layout or connection, then volume **1103** sends the message respectively to movies list **1105**, menus list **1106** or connections list **1107**. Each of movies list **1105**, menus list **1106** and connection sets list **1107** operates similarly to objects described thus far. Each parses through a received message for included control information, sends the message respectively to a corresponding movie object, menu object or connections list and adjusts its totals variable as needed.

A movie message, for example, will then progress down through the movie object tree, and, depending upon the message type, will be filtered, by track list **1152**, track-**1** **1153***a* and then handled a matching clip, or will be filtered by chapter list **1152** and then handled by corresponding chapter or by a clip properties object (i.e. as illustrated). Menu layout data will similarly progress (as illustrated) down through the menus list tree, being handled by a matching menu properties object, and connections data will progress down the connection sets list tree until it is handled by a connection object (with reference to its source pointer or destination pointer variables). Upon receipt, a clip properties, menu key, end key, menu properties or connection object will handle the message and store included data, delete stored data or modify stored data in a similar manner as with media database object **1103**.

Each respective storage object stores authoring modifications in a sequentially indexed list according to its type (i.e. each object name is illustrated to reflect the data type the object stores). Thus, for example, chapter points within a movie are stored from a first chapter point during playback to a final chapter point in the movie. (Playback will however, be determined by authored connections.) The list accommodates added, inserted or deleted data interactively by expanding or contracting about the addition, insertion or deletion point.

While other data structures might be utilized, interactively adjusted indexed lists and limited object definitions, using even a minimally equipped computer, are sufficiently robust to accommodate an author's input rate, given the relatively small amount of data stored in each list. Alternative structures that might be used, for example, include but are not limited to a lesser number of objects each containing a less restricted dataset and/or the addition of summary objects for storing total numbers of menus, buttons and system other status and/or statistical information. Such arrangements however, have been found to add complexity with only moderate gains in application-specific operational characteristics. Alternative data structures, including but not limited to multi-dimensional arrays, multiple queues and linked lists stored locally and/or remotely, present similar tradeoffs.

Data management engine **165** returns stored data to authoring engine **160** in a manner essentially the reverse of that for storing data. Volume **1100**, upon receipt of a request for stored data, parses the request call for a data type, searches its contained objects list for a corresponding object, and forwards the request to title key jump source **1101**, first play jump source **1102**, media database **1103**, DVD layout properties list **1105**, movies list **1106**, menus list **1107** or connection sets list **1107**. Movies list **1105**, menus list **1106** or connection sets list **1107**, upon receipt of such a request, parses its to available objects list and forwards the message correspondingly to a movie object, menu object or connection list object, and so on, until the message is received by a last recipient object. The last recipient object then retrieves the requested data and sends the data in the reverse direction of request receipt until the data reaches volume **100**. Volume

1100, upon receipt of the data, sends the requested data to authoring engine 160. (Error handling and messaging functionality are otherwise handled in a conventional manner.)

Data management engine 165 further responds to queries from authoring engine 160 for purposes such as totaling the number of data elements of a given type or for reviewing the contents of a particular object's data list. As with data storage and retrieval above, data management engine 165 receives a call from authoring engine 160 requesting information. Volume 1100 parses the message, polls its available objects list and sends the message to a corresponding object. For objects linked to a tree-structure, such as movies list 1105, menus list 1106 and connection sets list 1107, the message is forwarded down through respective objects as already discussed, and a last recipient object will respond. If the message requests, for example, a total number of data elements of a given type, then a last recipient will either poll its totals variable or, if necessary, poll its data list for corresponding data, count the number of corresponding occurrences and return a response including the total. The response is sent back through the tree structure to volume 1100, which sends the message (including the total) to authoring engine 160. Given the relatively small number of objects, alternatives (such as asynchronous multiple-messaging and, in particular, broadcast messages) add some expediency, but with unnecessarily added complexity.

As with the authoring engine interface objects, the object types, inter-object messaging protocol and data objects utilized in data management engine 165, in view of the disclosure herein, will be apparent to those skilled in the computer arts. Preferably, available object libraries from Microsoft® are utilized. For example, the preferred available objects and data lists utilize Standard Template Libraries and, in particular, Expandable Indexed Buffered/Vectored Lists. Such objects provide robust response with the flexibility of expandable lists and indexed vectors for easy lookup in light of the typically small number of objects and datasets, among other factors. As noted earlier however, use of an object-oriented architecture and/or the specific data structures are not essential and many conventional alternatives can be utilized.

As discussed, the particular arrangement of objects of the preferred data management engine 165 is preferred according to its flexibility, performance and adaptability among other factors. It should be noted therefore, that any number of modifications will be apparent according to the teachings and within the spirit and scope of the invention.

FIGS. 12*a* through 15, with reference to FIGS. 2 and 11, illustrate compilation according to a preferred embodiment of the invention.

As shown generally in FIG. 12*a*, compiler 170 (FIG. 2) preferably operates on data entered through the authoring process into the interface 160*a* of authoring engine 160 (FIG. 2) and stored by data management engine 165 in three stages. In step 1201, compiler 170 builds an intermediate skeleton-form PGC layout data structure. The skeleton-form PGC layout data structure is preferably formed according to DVD program code segment ("program chain" or "PGC") abstractions and a network abstraction according to the invention, utilizing only summary data gathered from data management engine 165. Broadly stated, each PGC abstraction is preferably comprised of pre-determined command combinations, wherein the number of PGCs of a given type and the number of command combinations of a given type (e.g. button command combinations) are determined according to either a default value (e.g. typically one PGC) or according to the number of corresponding authored element types. (e.g. the number of menu buttons in a given menu).

In step 1203, compiler 170 resolves source-target connections as indices to source and target identifier information within data management engine 165. In step 1205, compiler 170 replaces the indices with identifier information which is retrieved by further querying data management engine 165.

FIG. 13 illustrates a preferred PGC layout structure according to the invention. As shown, the PGC layout structure is divided into a single first play PGC space 1301 (in accordance with the DVD Specification), a single video manager ("VMGM") domain 1302, and one or more video title set ("VTS") domains (e.g. 1303 and 1304) according to the number of movies in the movie title.

The preferred VMGM domain PGC layout structure includes a single title key PGC abstraction, 1321 and a single movie router PGC abstraction 1322. Thereafter, the VMGM PGC structure includes 2 menu PGC abstractions (e.g. 1323*a* and 1323*b*) for each authored menu and a single PGC abstraction for each end command (in each movie) that an author for which an author has specified a connection. As will be discussed further, each menu PGC abstraction pair includes a menu display PGC (e.g. 1323*a* and 1324*a*) and a menu button router PGC (e.g. 1323*b* and 1324*b*).

Each VTS domain PGC layout structure (e.g. 1303) includes a movie display PGC 1331 and a video title set menu ("VTSM") area 1332. VTSM area further consists of from one to four remote key router PGCs (e.g. remote key router PGCs 1332), depending upon the number of different remote key commands necessary, given the preferred layout structure, to realize the chapter target connections selected using connection view 901. More specifically:

number of remote key router PGCs in a given VTSM=total number of chapter points in a corresponding movie/25 (rounded, if a non-integer, to a next higher integer value).

In each case, an attempt has been made to minimize the number of PGCs without detrimental impact on flexibility. Thus, while the number of PGCs is as indicated above, complete authoring flexibility with regard to connecting menus, menu buttons and presentation data without concern for limitations of the DVD programming specification 207 (FIG. 2) is provided. Further, the practical impact of resultant limitations is also minimized.

For example, the number of remote key router PGCs per VTSM area calculation reflects that each chapter point abstraction requires more than four commands. This in turn reflects that only one hundred twenty eight commands are allowable in a single PGC chain in accordance with the DVD programming specification 207. While not essential, placing each abstraction completely within a separated chains and in equal numbers throughout like chains provides an efficiently symmetrical structure. Since DVD programming specifications 207 provide for up to ninety nine chapter points per movie, a maximum of four PGC abstractions is required without detrimental impact in terms of connectability. Considering the same parameters and calculations for menus however, it is seen that only twenty five menu buttons are available per menu without limitation on connectability. In practical terms however (i.e. displaying a menu on a conventional television set), this number does not present any practical detrimental effect.

The use of consecutive locations in the PGC layout structure greatly simplifies the task of finding specific PGCs relating to specific data types and further for resolving PGC

connections. A movie title PGC will always be the first element, a movie router PGC will always be the second element, and a display menu PGC can always be located merely by adding a known constant plus two times the menu number, etc.

Those skilled in the art will appreciate however, in view of the discussion herein, that the PGC abstractions provide for other than consecutively arranged elements as an indexed list in memory **120** (FIG. **1**). Such alternatives, for example, include but are not limited to multiple lists, queues and/or multi-dimensional arrays stored in memory, in other media, and/or in more than one media either locally or in a distributed fashion, as with data management engine **165**. Such methods can be useful where more than one authoring location or other distributed environments are utilized.

The FIG. **12***b* flowchart, with reference to FIG. **13**, shows in greater detail how compiler **170** constructs a preferred PGC layout data structure in an initial skeleton form. As shown, compiler **170** begins by storing a first play PGC abstraction, a title key PGC abstraction and a menu router PGC abstraction into PGC layout structure **1300** (FIG. **13**) in steps **1207**, **1208** and **1209** respectively. Next, in step **1213**, compiler **170** queries data management engine **165** for a total number, MenusTot, of menus authored and, in step **1214**, initializes a menu pointer, MenuPtr. In step **1215**, compiler **170** queries data management engine **165** for a total number, ButtonsTot, of buttons authored in a current menu (e.g. initially, a first menu). MenusTot will specify the number of predetermined menu display and menu button router PGC abstractions (i.e. "menu PGC abstraction pairs") that compiler **170** will add to the structure, while ButtonsTot will specify the number of commands that compiler **170** will add to each PGC of a current menu PGC abstraction pair.

In step **1216**, compiler **170** adds a menu PGC abstraction pair to VMGM PGC structure **1302** (FIG. **13**) corresponding to the existence of and the number of buttons in a current authored menu (e.g. initially, a first menu). If, in step **1217**, one or more menus are not yet added to VMGM PGC structure **1301**, then in step **1218**, compiler **170** increments the menu counter and returns to step **1211**.

At this point, compiler **170** lacks any authoring information other than MenusTot and a respective ButtonsTot value for each current menu. A similar same lack of further authoring details will also exist for other PGCs in the skeleton-form, PGC layout structure. The preferred PGC and network abstractions of the invention however, enable compiler **170** to accommodate missing authoring details merely by inserting null values ("no-ops") into the commands of the abstracted PGCs for unknown connection information (i.e. source- target identification information). As discussed, compiler **170** will preferably resolve these no-ops later in compilation. These abstractions further enable menu PGCs to be created independently of movies and movie arrangements. Thus, independently created/conceived menu PGCs provide extensive flexibility, allowing an author to link any available menu button of any menu to any potential target using a user-friendly interface such as the preferred connection view **901**.

If instead, in step **1217**, all authored menu layouts are reflected by corresponding menu PGC abstraction pairs, then compiler **170** proceeds to step **1219**. In step **1219**, compiler **170** queries data management engine **165** for the total number, MovieTot, of movies, which compiler **170** will use to create end commands, VTSs and VTS contents. In step **1221**, compiler **170** initializes a current movie pointer ("MoviePtr"), as well as two counters, "EndTot" and "Remote". Compiler **170** will use EndTot to count the number of available end-of-chapter conditions in each movie for which an author has specified connections and will use Remote to count the number of available playback interruption conditions (i.e. by a user pressing a DVD-player control, typically on a remote control device) for which an author has specified connections.

In step **1223**, compiler **170** queries data management engine **165** for the total number of chapters ("ChapterTot") in a current movie (e.g. initially, the first movie) and, in step **1225**, initializes a current chapter pointer ("ChapterPtr"). If, in step **1227**, the author has specified a target for the current chapter, end-of-chapter condition (i.e. using connection view **901**), then, in step **1229**, compiler **170** increments EndTot; otherwise, compiler **170** proceeds to step **1231**. Similarly, if, in step **1231**, the author has specified a target for the current chapter, remote-control key playback interruption ("remote-key") condition, then, in step **1233**, compiler increments Remote; otherwise, compiler **170** proceeds to step **1235**.

The existence of authored connections is determined similarly for both end-of-chapter and remote-key conditions. Preferably, objects **1101–1163** (FIG. **11**) contain actual source and target identifier information (i.e. corresponding to authored sources and targets), while the connection objects (e.g. **1171***a*) contain pointers to data stored by these objects. Stated alternatively, as a new potential source is authored, a connection object is instantiated, including a source pointer that points to the potential source and a null-value target pointer; if an author later connects such a source, then the correspondihg connection-object target pointer value is replaced by a pointer to the target object. (Subsequent editing by an author correspondingly deletes or instantiates a connection object and/or changes a source pointer or target pointer value.)

Therefore, compiler **170** determines the existence of a connected end command by first querying each connection object for a source pointer pointing to the currently selected chapter-object. Once found, compiler **170** checks the corresponding target pointer. A null-value target pointer indicates an unconnected end command while a non-null-value target pointer indicates the existence of a connection. Remote key (i.e. "menu key" in FIG. **11**) connections are similarly determined by finding an identifier in a current chapter menu key object (e.g. **1157**), finding the corresponding source pointer in one of the connection objects, and then querying the connection object for the existence of a corresponding non-null-value target pointer.

Those skilled in the art, in view of the foregoing, will appreciate that considerable variation of the above structure will provide the same, related or similar functionality. For example, identifiers, labels and even complete movie tree, menu tree and/or other objects could well be contained within or duplicated within the connections-tree (i.e. objects **1107–1173**). A single connection object could also be used (i.e. having a single list of all connections), as could connection objects that remain despite the deletion of a source. Other variations are also anticipated. The current structure is however, preferred in that it provides a compilation time of only a few milliseconds, minimizes memory usage and further facilitates debugging, emulation, simulation and overall symmetry by separating these objects (and their contained data). In simulation, for example, the restrictions imposed by the DVD Specification are not controlling and simulation can therefore more efficiently utilize authoring data directly from the preferred, non-integrated data management engine **165** object structure.

Returning now to FIG. **12***b*, if, in step **1235**, more chapters remain in the current movie, then compiler **170** increments

ChapterPtr and returns to step **1227**; otherwise, compiler **170** proceeds to step **1237**. In step **1237**, compiler **170** adds a 1–4 PGC, end command router PGC abstraction to layout structure **1300** (FIG. **13**). In step **1238**, compiler **170** creates a VTS domain for the current movie (i.e. including a VTSM), adding to the VTS domain a movie display PGC in step **1239** and adding a 1–4 PGC, remote key PGC abstraction in step **1240**.

If, in step **1241**, more movies remain in the current movie title (i.e. tested by comparing MovieTot with MoviePtr), then compiler increments moviePtr in step **1243**, re-initializes EndTot and Remote in step **1245** and returns to step **1223**. Otherwise, formation of a PGC layout structure in skeleton form has been completed.

The FIG. **12***c* flowchart with reference to FIG. **11** shows how compiler **170** replaces the no-ops in (skeleton form) PGC layout structure **1300** with indices (i.e. source or target pointers) to respective sources and targets, and then further replaces the indices with element identifiers. In step **1251**, compiler **170** initializes a movie pointer ("MoviePtr") to a first movie, a chapter pointer ("ChapterPtr") to a first chapter, a menu pointer ("MenuPtr") to a first menu and a button pointer ("ButtonPtr") to a first button.

In step **1253**, compiler **170** queries data management engine **165** (i.e. connection-objects) for a source-pointer to a next (initially, a first) author-connected button. As discussed earlier, the connection object checks its source-pointer for a corresponding source having a corresponding non-null-value target pointer. Since specific connection values (rather than the existence of a connection as with FIG. **12***b*) are required in this case, the query utilized results in the return of such a source-pointer. In step **1255**, compiler **170** uses the returned source-pointer to query data management engine **165** for the corresponding target-pointer and, in step **1257**, compiler **170** uses the returned indices to query data management engine **165** (e.g. via volume **1100**, menu- **1** **1160***a* and button list **1161** to button-1 **1162***a*) for the source and target identifiers corresponding to the source and target pointers. Then, in step **1259**, compiler **170** replaces the current button command no-ops (of the current menu PGC abstraction pair) with the returned identifiers.

If, in step **1261**, more buttons remain unresolved in the current menu, then compiler **170** increments ButtonPtr in step **1263** and returns to step **1253**; otherwise, compiler **170** proceeds to step **1265**. If, in step **1265**, menus remain unresolved, then compiler **170** increments MenuPtr and resets ButtonPtr to one in step **1267**, and then returns to step **1253**; otherwise, compiler **170** proceeds to step **1271**.

Having resolved and replaced all menu button no-ops, compiler **170** next resolves all chapter end-command and remote-key PGC abstraction no-ops in a similar manner. Compiler **170** queries data management engine **165** for a (next connected) current chapter end command source-pointer in step **1271**, uses the returned source-pointer to query data management engine **165** for a corresponding target-pointer in step **1272**, uses the pointers to query data management engine **165** for corresponding identifiers in step **1273** and replaces corresponding layout structure **1300** PGC commands with the returned identifiers in step **1274**. Similarly, compiler **170** queries data management engine **165** for a (next connected) current remote key source-pointer in step **1277**, uses the returned source-pointer to query data management .engine **165** for a corresponding target-pointer in step **1278**, uses the pointers to query data management engine **165** for corresponding identifiers in step **1279** and replaces corresponding layout structure **1300** PGC commands with the returned identifiers in step **1280**.

If, in step **1283**, more chapters remain unresolved, then compiler **170** increments the chapter pointer in step **1285** and returns to step **1271**. If instead, no chapters remain unresolved in the current movie, then compiler **170** proceeds to step **1286**. In step **1286**, compiler **170** queries data management engine **165** (i.e. via volume 1100 to media database **1103** of FIG. **11**) for all audio and video file references which reference the current movie. In step **1287**, compiler **170** invokes multiplexer **185**, which retrieves the referenced audio and video files and outputs a resultant multiplexed data file in a conventional manner and in accordance with the DVD disk format specifications **205** (FIG. **2**) of the DVD Specification.

If, in step **1288** more movies remain unresolved in layout structure **1300**, then compiler **170** resets pointers for the next movie and first chapter in step **1289** and returns to step **1271**. Otherwise, compiler **170** (in a similar manner) resolves first play, title key jump source and menu router no-ops respectively in steps **1291**, **1293** and **1295**. Then, in step **1297**, compiler **170** saves the PGC layout structure as a stored file.

With regard to FIG. **12***b* and **12***c*, total authored element values (i.e. such as MenusTot and ButtonsTot) are maintained on an ongoing basis in a corresponding list object or the functional equivalent of a list object as already discussed. For example, movies-list object **1105** (FIG. **11**), in addition to a list for containing references to all instantiated movie objects, also contains a variable for updating the total number of movies in a current movie title during the course of one or more authoring sessions. Similarly, button-list object **1161** contains a list of instantiated button objects (e.g. **1162***a* through **1162***b*) as well as a variable indicating the total number of buttons in menu-1. Other list objects similarly include ongoing totals which are updated during the course of authoring. One reason is that some early-generation DVD-players limit the available memory space for storing PGCs, which correspondingly limits the number of elements (e.g. menus, menu buttons and chapters) that the invention permits to be authored. These limits and/or current totals are therefore selectively conveyed to an author through interface **160***a*. Ongoing totals are also beneficial in that no time periods are required during compilation for calculating such totals.

As will be understood by those skilled in the art however, total values might become unimportant for other than compilation purposes as DVD-players are manufactured with increasing resources in conformance with the current DVD Specification, in accordance with expanded DVD capabilities and in accordance with the requirements of non-DVD systems. In such cases, totals can alternatively be calculated during compilation.

The use of preferably pre-determined PGC abstraction types comprising preferably pre-determined command combinations and the preferred PGC layout structure are thus factors in providing a maximized authoring flexibility and efficient compilation among other benefits. Available connections remain completely flexible during authoring and, in fact, until substitutions are made for no-ops during compilation. The preferred structures of PGC abstractions further add to compilation efficiency, since a skeleton can be formed with only summary authoring data, and then authoring details can be quickly added thereafter.

FIGS. **14** and **15**, with reference to FIG. **13**, illustrate a preferred network or "connection-switching" abstraction according to the invention. The connection-switching abstraction, while operationally active only during playback of a movie-title, is also a factor in determining PGC abstractions produced by compiler **170** as well as the movie, menu

and connection movie-title abstraction utilized by data management engine **165**, interface **160***a* and authoring engine **160** (FIG. **2**).

Details of the DVD Specification including but not limited to multiplexed data stream and DVD player configurations, data formats, protocols and loading of data are known to those skilled in the art and will therefore be discussed only to the extent required for an understanding of the invention.

DVD programming specifications **207** (FIG. **2**) provide that PGCs can reside (along with the corresponding presentation data) in virtual structures including a first play space, a video manager ("VMGM") and any of 99 video title sets ("VTSs"), each of which includes a video title set menu space ("VTSM"). Among the limitations of this virtual structure however, is first that a PGC in an initial VTS or VTSM cannot directly trigger (i.e. jump to, using a DVD jump command) a PGC stored in another VTS (or VTSM). For example, while a PGC in an initial VTS can "playback a chapter of presentation data" and the conclusion of chapter playback can trigger a "followup" PGC, the followup PGC cannot be stored in a different VTS. Similarly, an initial PGC used to respond to DVD consumer menu-button activation cannot trigger a second PGC which is stored in a different VTS. A further relevant limitation is that the format of performance data must remain constant within a given VTS. So, for example, a video data stream having one aspect ratio cannot be stored in the same VTS with another video data stream having a different aspect ratio.

The FIG. **14** functional diagram illustrates how the preferred connection-switching abstraction provides a flexible and robust functional superstructure within which movie-title, DVD-player and interactively occurring consumer-control events are routed and executed. In the figure, VTS-A **1303** and VTS-A+1 exemplify any two different VTSs which have been created during compilation of a movie-title. It should also be noted that the illustrated connection arrows only denote the "path" from one box (i.e. PGC abstraction, PGC or command-set) to another that can result from an author's use of connection view **901** (FIG. **9**). Thus, fewer connections than those illustrated might be authored and each path from one box to another is accomplished individually using a single "jump command" or a single transfer of control by a DVD-player. (The use of multiple connected arrows and shared arrows is used only for clarity sake, since the alternative use of individual arrows between each pair of boxes might otherwise obscure the invention.)

Within each VTS, only a movie display PGC abstraction operates as a "control PGC" (i.e. directly controls menu and/or movie display). For example, VTS-A **1303** includes movie display PGC abstraction **1331** and (within its VTSM domain **1322**) remote key PGC abstraction **1322***a*. Movie display PGC abstraction **1331** comprises a single PGC which includes a command-set ("pre-command") for selecting a chapter and initiating playback of the chapter, as well as an end command "cell command" that initiates routing upon the occurrence of an end-of-chapter-playback condition. Remote menu key **1431***a*, which denotes an automatic DVD player function, traps and forwards a remote-key condition (i.e. user depression of a remote menu key which interrupts playback). Remote menu key router PGC abstraction **1322***a* of VTSM-A **1322** sets the authored target for a corresponding remote menu key condition (i.e. where a consumer presses a remote menu key during playback) and then routes control to a corresponding movie PGC abstraction or menu PGC abstraction within VMGM **1302**. Other VTSs (e.g. VTS-A+1 **1304**) are similarly structured for each movie within the current DVD movie-title.

Each remote menu key router PGC abstraction includes up to 4 PGCs to accommodate the up to 99 chapter points per movie limitation of the DVD Specification. The first remote menu key PGC is always assigned as a root menu and is always a hardwired (i.e. unalterable) target for any remote menu key condition (in accordance with the DVD Specification). Therefore, in order to provide for chapter dependent routing of a remote menu key condition, a DVD-player system register must first be queried for the last played chapter. Using the returned last played chapter information, program execution is then diverted to the corresponding authored remote menu key router PGC.

VTSM **1302** comprises the discussed menu display PGC (e.g. **1322**) and menu button router PGC (e.g. **1323***b*) abstraction pairs (for providing menu control), as well as the remaining router PGC abstractions. More specifically, movie router PGC abstraction **1322** acts as a playback bridge between VTS domains, receiving control from a remote key PGC in a first VTS (e.g. remote key PGC **1322***a* of VTS **1303**) and then forwarding control to a movie display PGC abstraction in second VTS (e.g. movie play PGC **1341** of VTS **1304**). In contrast, end router PGC abstractions (e.g. **1325** and **1326**) can be author-connected to route control from an end-of-chapter condition to either a selected chapter in a selected movie, or to a selected menu button in a selected menu.

As shown, a separate PGC is provided for each author-connected end-of-chapter condition. Each end command router PGC abstraction is paired with (i.e. responds to) a specific end command such that each end-of-chapter condition for a given movie will be routed from the end command to a unique end router PGC abstraction. Separate end command PGCs are required due to a flaw in current generation DVD-players whereby the last played chapter is not reliably available at the end of chapter playback. Upon correction of this flaw in future generation DVD-players however, end command routing can be accomplished in a manner consistent with remote menu key PGC abstractions (i.e. using only up to four end-command router PGCs per movie).

A menu display PGC abstraction (e.g. **1323***a*), when it receives control as a target and thereafter while a consumer continues to depress menu navigation buttons, effectuates control by highlighting a menu button and displaying the menu. If however, a consumer activates a menu button, then the DVD-player initiates the corresponding router PGC abstraction (e.g. **1323***b*), which routes control (i.e. according to an authored connection) to either a movie display PGC or to a menu display PGC.

For clarity sake, the first play PGC abstraction **1301** and title key PGC abstraction **1321** (FIG. **13**) are not shown in FIG. **14**. Each operates to transfer control to either a menu display PGC or a movie display PGC as with the end command router PGCs and menu router PGCs. First play PGC **1301** is stored in a separate DVD-player storage location, while title key PGC **1321** is stored in VMGM **1302**.

While those skilled in the art will appreciate, in view of the discussion herein, that considerable variation might be utilized, iterative experimentation with different connection-switching abstractions and DVD players has revealed a number of considerations. For example, command execution delays will necessarily occur as a result of PGC execution and greater delays typically result from transfer of control between a VTS (e.g. **1303** and **1304**) and VMGM **1302**. Another example is that a delay occurring prior to the start of a movie is observed to be more acceptable than a similar delay during navigation through what can be a large number

of menus. A still further example is that consistent delay periods for similar transitions is more acceptable than inconsistent delays for similar transitions.

Thus, the preferred connection-switching abstraction provides a generally symmetrical structure wherein delays are first minimized by source-router-target execution paths having a minimum number of PGCs and PGC commands. Movie display PGC abstractions are further placed similarly within each VTS, while menu PGC abstraction pairs are placed similarly within VMGM **1302**. (Note that an author typically only connects the end command of a last chapter within any given movie, such that the DVD-player will continuously play all chapters with the movie before control is routed outside the corresponding VTS). In addition, movie router **1322** is only used for VTS-to-VTS transitions This reflects, for example, that inconsistent delay between movie-to-movie playback and menu-to-movie playback is more acceptable than imposing further delay on menu-to-movie playback or other alternatives. (For example, further distribution and/or re-distribution of movie and/or menu routing functions have been observed to produce subjectively less acceptable results.) In addition, movie router **1322** complexity and PGC length is therefore reduced. It should be understood however, that these already short delay periods will further decrease as advances are made in DVD-player technology and that the resulting decreasing importance of such considerations might well contribute to further connection-switching abstraction variations.

The FIG. **15** flowchart broadly illustrates the operation of preferred connection-switching abstraction **1400**. In step **1503**, first play PGC abstraction is invoked in response to insertion of a movie-title into a DVD-player. The first play PGC abstraction (i.e. now the current PGC abstraction) determines target information (i.e. a target identifier and, if needed, target parameters). If, in step **1505**, a router is required, then, the current PGC abstraction routes the target information and control to a next router abstraction in step **1507** and operation returns to step **1511**. If no router is required in step **1505**, then, in step **1509**, the current PGC abstraction routes the target information to the target PGC abstraction.

If, in step **1511**, the target is not a chapter (i.e. playback of a chapter is not the resultant authored event) then the target displays a menu (i.e. according to the target information) in step **1513** and the DVD-player waits for a menu button to be selected (i.e. step **1513** through **1515** act as a wait loop). If, in step **1515** a menu button is selected, then the current PGC abstraction sets authored target information for the selected button in step **1517** and operation returns to step **1505**.

If instead, in step **1511**, the target is a chapter, then the target initiates playback of the chapter. If further, in step **1525**, a consumer invokes the remote menu key during playback of the chapter, then the current PGC abstraction sets authored target information in step **1527** and operation returns to step **1505**. If, in step **1525**, the remote menu key is not invoked (i.e. the chapter plays uninterrupted to its conclusion) and a chapter end command target has been authored, then the current PGC abstraction sets the authored target information in step **1537** and operation returns to step **1505**. If, in step **1535**, a chapter end command target has not been authored, then operation continues in step **1545**.

If, in step **1545**, more chapters exist in the current movie, then the DVD player increments the chapter number in step **1543** and operation returns to step **1523**. If instead, in step **1545**, no more chapters remain unplayed in the current movie, then the player suspends playback and (in some models) switches itself off.

For clarity sake, the operation of preferred connection-switching abstraction **1400** will also be discussed, by way of example, with reference to FIG. **14**. If, for example, an authored-connection for first play is set to begin playback of a first chapter of a first movie stored in VTS-A **1303**, then upon insertion of the DVD movie-title into a DVD-player, movie display PGC abstraction **1331** will be invoked. Movie display PGC **1331** will select and initiate playback of the first chapter.

If the first chapter playback is interrupted by a remote menu key condition, then the DVD-player will automatically trap the condition (i.e. box **1431***a*) and will initiate the root menu PGC of remote menu key router **1322***a* of VTSM-A **1322**. Assuming further that less than 25 chapters exist in the first movie, the root menu PGC of remote menu key router **1322***a* (i.e. now the current source PGC abstraction) will set the author-selected target for the first chapter remote menu key condition and will route control to either movie router **1322** or a menu display PGC (e.g. **1323** or **1324**) within VMGM **1302**. If movie router **1322** receives control, then upon receipt, movie router further routes control to the author-connected movie display PGC, in this case, movie display PGC **1341** of VTS-A+1 **1304**, which will set and initiates playback of the author-selected chapter of the VTS-A+1 movie.

If instead, playback of the first movie is not interrupted and only the last chapter of the first movie includes an author-connected end command, then the DVD-player will continue to play successive chapters of the first movie until the conclusion of the last movie. At the conclusion of the last movie, the DVD-player will execute cell command **1431***b* (i.e. end command), which will transfer control to the PGC in end router **1325** (in VTSM **1302**) that corresponds with the chapter last chapter played, i.e. the last chapter of VTS-A movie. (Since, in this case, only one chapter in the VTS-A movie has a connected end-of-chapter playback condition, end router **1325** will include only the one corresponding PGC.)

Upon receipt of control from end command **1431**, end router **1325** (i.e. now the current source) will set the corresponding author-connected target included in end router **1325**. Assuming the target is the VTS-A+1 movie, end router **1325** will further route control to movie display PGC **1341** of VTS-A+1 **1304**, which will set and initiate playback according to the chapter of the VTS-A+1 movie set by end router **1325**. (Since control is not being routed from one VTS to another VTS, movie router **1322** is not utilized.)

If instead, the current source PGC of end router **1325** (i.e. again, the only PGC in end router **1325** in this example) includes an author-selected connection to menu N **1323**, then end router **1325** will set target parameters and will route control to menu display PGC **1323***a*. Menu display PGC **1323***a* will highlight the button of menu-N **1323** according to the received target parameters and will then display menu-N **1323**. Menu display PGC **1323***a* will thereafter continue to be invoked by the DVD-player and will continue to highlight a button and display menu-N **1323** correspondingly with each successive uninterrupted (i.e. by consumer selection of a conflicting DVD control function) consumer depression of a navigation button. If however, the consumer next activates a displayed menu button, then the DVD-player will invoke menu button router PGC **1323***b*. Once invoked, menu button router PGC **1323***b* will set target parameters according to the author-selected connection for the activated button, and so on.

Attachment A attached hereto provides computer listings of preferred PGC abstractions source code according to the

invention. For clarity sake, compilation has already been completed. Stated alternatively, the no-ops initially included in the skeleton-form PGC layout structure have been replaced by indices and the indices have been resolved to source and target identifiers using the discussed compiler and compilation methods.

As shown in attachment A, the preferred PGC abstractions utilize a number of DVD player registers. According to the DVD specification, each DVD player includes 16 general purpose registers ("GPs"), and 20 system registers ("SPs"). The GPs are functionally undefined and merely "available for use" by movie title control program PGCs. Conversely, the SPs have fully defined purposes consistent with DVD player operation and movie title control program interfacing.

The preferred GPs utilization and corresponding naming conventions according to the invention are indicated in the following chart. As shown, PGC abstractions exclusively utilize only 5 GPs, leaving a maximized number of remaining GPs available for adding further capabilities.

Register	Referenced as	Description
GP10	Stream Select	Bit 15 = Select audio stream on/off Bit 14 = Select subtitle stream on/off Bit 13 = Select angle stream on,off Bits 10–12 = Audio stream number Bits 7–9 = Angle stream number Bits 0–6 = Subtitle stream number
GP12	Target Movie Number	Stored number = Movie number
GP13	Target Button Number	Stored number = Button number
GP14	Target Chapter Number	Stored number = Chapter number
GP15	Temporary Register	Stored number = value used with current PGC
SP7	Last Chapter Played	DVD player fills the register with the number of the last chapter played
SP8	Last Highlighted Button	DVD player fills the register with the number of the last highlighted button

As illustrated by the register utilization chart, GPs are utilized by source PGC abstractions primarily for designating (i.e. resolving an available connection to) target PGC abstractions and for passing to the targets parameters affecting target operation. The GPs are further utilized by target PGC abstractions primarily for establishing, manipulating and recalling localized variables (i.e. relating to a currently executing PGC command set).

For example, at a time prior to initiating playback of a chapter, a source PGC abstraction stores a value in GP10 ("stream select"). That value will later indicate to a target PGC which audio, subtitle and/or angle stream is to be selected for movie playback. A further example is that, at a time prior to routing control to a target PGC abstraction, a source PGC abstraction stores a target's designation in a combination of registers GP12 ("Movie Number") and GP14 ("Chapter Number") for a movie target or GP13 ("Button Number") for a menu target. Finally, PGC abstractions preferably utilize GP15 to temporarily store values, typically for use within a current PGC operation.

In most cases, only a portion of a given register ("register bits") are utilized, while conversely, a given register may be used for multiple purposes, as seen in the utilization of GP10 in the register chart. Those skilled in the art will appreciate, given the discussion herein, that the preferred embodiment enables certain advantages. Among these are that a single register or register set can be designated in all cases for similar purposes, thereby minimizing complexities, the number of registers required and the number of commands required within a PGC without detrimentally affecting routing or parameter passing flexibility. Similarly, operations required to parse register data containing multiple data values are not needed. Other arrangements consistent with the teachings of the invention however, are likely in view of other applications facilitated by these teachings and in accordance with the scope and spirit of the invention.

While the present invention has been described herein with reference to a particular embodiment thereof, a latitude of modification, various changes and Substitutions are intended in the foregoing disclosure, and it will be appreciated that in some instances some features of the invention will be employed without a corresponding use of other features without departing from the spirit and scope of the invention as set forth.

APPENDIX A: PGC ABSTRACTION SOURCE CODE

Exemplary Included Movies/Menus

2 Movies with 3 chapters each
2 Menus with 4 Buttons each
Exemplary Included Connections

Connections Source		Connections Target
Movie 1 : Chapter 1 Remote Key	->	Menu 1: Button 1
Movie 1 : Chapter 2 Remote Key	->	Menu 1: Button 2
Movie 1 : Chapter 3 Remote Key	->	Menu 1: Button 3
Movie 1 : Chapter 3 End	->	Menu 1: Button 1
Movie 2 : Chapter 1 Remote Key	->	Menu 2: Button 1
Movie 2 : Chapter 2 Remote Key	->	Menu 2: Button 2
Movie 2 : Chapter 3 Remote Key	->	Menu 2: Button 3
Movie 2 : Chapter 3 End	->	Menu 2: Button 1
Menu 1 : Button 1	->	Movie 1: Chapter 1
Menu 1 : Button 2	->	Movie 1: Chapter 2
Menu 1 : Button 3	->	Movie 1: Chapter 3
Menu 1 : Button 4	->	Menu 2: Button 1
Menu 2 : Button 1	->	Movie 2: Chapter 1
Menu 2 : Button 2	->	Movie 2: Chapter 2
Menu 2 : Button 3	->	Movie 2: Chapter 3
Menu 2 : Button 4	->	Menu 1: Button 1

-continued

APPENDIX A: PGC ABSTRACTION SOURCE CODE

PGC ABSTRACTION SOURCE CODE

```
1. VIDEO MANAGER ("VMGM") Program Chains ("PGCs")
First Play PGC

PRE_CMD#1:   MovI   GP14,   1                          // Target = Chapter 1
PRE_CMD#2:   MovI   GP12,   1                          // Target = Movie 1
PRE_CMD#3:   JumpSS         VMGM PGCI 2                // Jump To Movie Router
Title PGC (PGC #1)

PRE_CMD#1:   MovI   GP13,   1                          // Target = Button 1
PRE_CMD#2:   LinkPGCN       PGCN = 3                   // Jump to Menu 1
Movie Router PGC (VMGM PGC #2)

PRE_CMD#1:   MovI   GP15,   1                          // Setup Comparision to 1
PRE_CMD#2:   EQ     GP15,   GP12    JumpTT TTN = 1     // If Target Movie = goto Movie1
PRE_CMD#3:   MovI   GP15,   2                          // Setup Comparision to 2
PRE_CMD#4:   EQ     GP15,   GP12    JumpTT TTN = 2     // If Target Movie = 2, goto Movie 2
PRE CMD#5:   JumpSS         First Play                 // Should never get here
Menu 1 Display PGC (VMGM PGC #3)

PRE_CMD#1:   Mov    GP15,   GP13                       // Put target button no. into temp storage
PRE_CMD#2:   MulI   GP15,   1024                       // Shift Button number by 10 Bits to the left
PRE CMD#3:   SetHL_BTNN     HLP = GP15                 // Highlight target button in menu; display menu
UZ,1/18 Menu 2 Display PGC (VMGM PGC #4)
PRE CMD#1:   Mov    GP15,   GP13                       // Put target button no. into temp storage
PRE CMD#2:   MulI   GP15,   1024                       // Shift Button number by 10 Bits to the left
PRE CMD#3:   SetHL_BTNN     HLP = GP15                 // Highlight target Button in menu; display menu
Menu 1 Button Router (VMGM PGC #5)

PRE_CMD#1:   Mov    GP15,   SP8                        // Put last highlighted button in temp storage
PRE_CMD#2:   DivI   GP15,   1024                       // Shift Button number 10 bits to the right
PRE_CMD#3:   NEI    GP15,   1 GoTo CMDNUM = 7          // If button number ≠ 1, goto CMD #7
PRE_CMD#4:   MovI   GP14,   1                          // Target = Chapter 1
PRE_CMD#5:   JumpTT         TTN = 1                    // Jump Movie 1
PRE_CMD#6:   Nop
PRE_CMD#7:   NEI    GP15,   2 GoTo CMDNUM = 11         // If button number ≠ 2, goto CMD #11
PRE_GMD#8:   MovI   GP14,   2                          // Target = Chapter 2
PRE_CMD#9:   JumpTT         TTN = 1                    // Jump Movie 1
PRE_CMD#10:  Nop
PRE_CMD#11:  NEI    GP15,   3 GoTo CMDNUM = 15         // If button number ≠ 3, goto CMD #15
PRE_CMD#12:  MovI   GP14,   3                          // Target = Chapter 3
PRE_CMD#13:  JumpTT         TTN = 1                    // Jump Movie 1
PRE_CMD#14:  Nop
PRE_CMD#15:  MovI   GP13,   1                          // Target = Button 1
PRE_CMD#16:  LinkPGCN       PGCN = 4                   // Jump Menu 2
Menu 1 Button Router (VMGM PGC #6)

PRE_CMD#1:   Mov    GP15,   SP8                        // Put last highlighted button in temp storage
PRE_CMD#2:   DivI   GP15,   1024                       // Shift Button number 10 bits to the right
PRE_CMD#3:   NEI    GP15,   1 GoTo CMDNUM = 7          // If button number ≠ 1, goto CMD #7
PRE_CMD#4:   MovI   GP14,   1                          // Target = Chapter 1
PRE_CMD#5:   JumpTT         TTN = 2                    // Jump Movie 2
PRE_CMD#6:   Nop
PRE_CMD#7:   NEI    GP15,   2 GoTo CMDNUM = 11         // If button number ≠ 2, goto CMD #11
PRE_CMD#8:   MovI   GP14,   2                          // Target = Chapter 2
PRE_CMD#9:   JumpTT         TTN = 2                    // Jump Movie 2
PRE_CMD#10:  Nop
PRE_CMD#11:  NEI    GP15,   3 GoTo CMDNUM = 15         // If button number ≠ 3, goto CMD #15
PRE_CMD#12:  MovI   GP14,   3                          // Target = Chapter 3
PRE_CMD#13:  JumpTT         TTN = 2                    // Jump Movie 2
PRE_CMD#14:  Nop
PRE_CMD#15:  MovI   GP13,   1                          // Target = Button 1
PRE_CMD#16:  LinkPGCN       PGCN = 3                   // Jump Menu 1
Title 1 End Router Chapter 3 (VMGM PGC #7)

PRE_CMD#1:   MovI   GP13,   1                          // Target = Button 1
PRE_CMD#2:   LinkPGCN       PGCN = 3                   // Jump Menu 1
Title 2 End Router Chapter 3 (VMGM PGC #8)

PRE_CMD#1:   MovI   GP13,   1                          // Target = Button 1
PRE_CMD#2:   LinkPGCN       PGCN = 4                   // Jump Menu 2
2. Video Title Segment #1 ("VTS-1") PGCs
Movie 1 Display PGC (VTS PGC #1)

PRE_CMD#1:   Mov    GP15,   GP14                       // Move Target Chapter into temp storage
PRE_CMD#2:   MovI   GP14,   0                          // Zero Target Chapter register
```

-continued

APPENDIX A: PGC ABSTRACTION SOURCE CODE

```
PRE_CMD#3:  EQI   GP15,  1   LinkPGN  PGN = 1  // If Chapter=1, Goto Program #1 (Chapter 1)
PRE_CMD#4:  EQI   GP15,  2   LinkPGN  PGN = 2  // If Chapter=2, Goto Program #2 (Chapter 2)
PRE_CMD#5:  EQI   GP15,  3   LinkPGN  PGN = 3  // If Chapter=3, Goto Program #3 (Chapter 3)
End CMD for Movie 1 Chapter 3 (VTS PGC #1)

C_CMD#1:    CallSS  VMGM_PGCN = 7, DOMAIND = 3   // Jump to VMGM PGC #7 (Title1, End-router for Chapter 3)
Chapter Router PGC (VTSM PGC #1)

PRE_CMD#1:  Mov   GP15,  SP7                     // Put last played chapter into temp storage
PRE_CMD#2:  Nop
PRE_CMD#3:  Nop
PRE_CMD#4:  Nop
PRE_CMD#5:  GEI   GP15,  2 GoTo CMDNUM = 9       // If last chapter ≧ 2, goto CMD #9
PRE_CMD#6:  MovI  GP13,  1                       // Target = Button 1
PRE_CMD#7:  JumpSS VMGM_PGCN = 3, DOMAIND = 3    // Jump to Menu 1
PRE_CMD#8:  Nop
PRE_CMD#9:  GEI   GP15,  3 GoTo CMDNUM = 13      // If last chapter ≧ 3, goto CMD #13
PRE_CMD#10: MovI  GP13,  2                       // Target = Button 2
PRE_CMD#11: JumpSS VMGM_PGCN = 3, DOMAIND = 3    // Jump to Menu 1
PRE_CMD#12: Nop
PRE_CMD#13: MovI  GP13,  3                       // Target = Button 3
PRE_CMD#14: JumpSS VMGM_PGCN = 3, DOMAIND = 3    // Jump to Menu 1
3. Video Title Segment #2 ("VTS-2") PGCS
PRE_CMD#1:  Mov   GP15,  GP14                    // Move Target Chapter into temp storage
PRE_CMD#2:  MovI  GP14,  0                       // Zero Target Chapter register
PRE_CMD#3:  EQI   GP15,  1   LinkPGN  PGN = 1  // If Chapter=1, Goto Program #1 (Chapter 1)
PRE_CMD#4:  EQI   GP15,  2   LinkPGN  PGN = 2  // If Chapter=2, Goto Program #2 (Chapter 2)
PRE_CMD#5:  EQI   GP15,  3   LinkPGN  PGN = 3  // If Chapter=3, Goto Program #3 (Chapter 3)
End CMD for Movie 2 Chapter 3 (VTS PGC #1)

C_CMD#1:    CallSS  VMGM_PGCN = 7, DOMAIND = 3   // Jump to VMGM PGC #8 (Title 2 End-router for Chapter 3)
Chapter Router PGC (VTSM PGC #1)

PRE_CMD#1:  Mov   GP15,  SP7                     // Put last played chapter into temp storage
PRE_CMD#2:  Nop
PRE_CMD#3:  Nop
PRE_CMD#4:  Nop
PRE_CMD#5:  GEI   GP15,  2 GoTo CMDNUM = 9       // If last chapter ≧ 2, goto CMD #9
PRE_CMD#6:  MovI  GP13,  1                       // Target = Button 1
PRE_CMD#7:  JumpSS VMGM_PGCN = 4, DOMAIND = 3    // Jump to Menu 1
PRE_CMD#8:  Nop
PRE_CMD#9:  GEI   GP15,  3 GoTo CMDNUM = 13      // If last chapter ≧ 3, goto CMD #13
PRE_CMD#10: MovI  GP13,  2                       // Target = Button 2
PRE_CMD#11: JumpSS VMGM_PGCN = 4, DOMAIND = 3    // Jump to Menu 2
PRE_CMD#12: Nop
PRE_CMD#13: MovI  GP13,  3                       // Target = Button 3
PRE_CMD#14: JumpSS VMGM_PGCN = 4, DOMAIND = 3    // Jump to Menu 2
```

We claim:

1. A method for compiling an authored DVD video program, the method comprising:

providing an abstraction layer between menu buttons, movie chapters and connections therebetween, and interconnected PGCs, their instructions and their allocation within a DVD video space and domain structure;

whereby an author of the DVD video program is able to author the DVD video program by referencing elements of the abstaction layer rather than by referencing the interconnected PGCs, their instructions and their allocation within the DVD video space and domain structure.

2. A system for compiling an authored multimedia presentation to form a DVD video program, the system comprising:

means for providing an abstraction layer on top of interconnected PGCs, their instructions and their allocation within a DVD video space and domain structure; and

means for compiling elements of the abstraction layer to generate DVD program code and content.

3. A system for compiling an authored multimedia presentation to form a DVD video program, the system comprising:

means for authoring DVD program content; and

means for interlinking content elements with one another using automatically generated dummy PGCs.

4. A method for compiling an authored multimedia presentation to form a DVD video program, the method comprising establishing an abstracted reference to program code referenced in a DVD program before an absolute reference to the program code is known.

* * * * *